uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Telemática
2016/2017

Trabajo Fin de Grado

Control de Congestión TCP y mecanismos AQM

Sergio Maeso Jiménez

Tutor/es
Celeste Campo Vázquez
Carlos García Rubio
Leganés, 2 de Octubre de 2017

Control de Congestión TCP y mecanismos AQM

By

Sergio Maeso Jiménez

Directed By

Celeste Campo Vázquez
Carlos García Rubio

A Dissertation Submitted to the
Department of Telematic Engineering
in Partial Fulfilment of the Requirements for the
BACHELOR'S DEGREE IN TELEMATICS ENGINEERING

Approved by the
Supervising Committee:

Chairman

Marta Portela García

Chair

Carlos Alario Hoyos

Secretary

Iñaki Úcar Marqués

Deputy

Javier Manuel Muñoz García

Grade:

Leganés, 2 de Octubre de 2017

Acknowledgements

I would like to thanks my tutors
Celeste Campo and Carlos Garcia
for all the support they gave me
while I was doing this thesis with them.

To my parents, who believe in me
against all odds.

Abstract

In recent years, the relevance of delay over throughput has been particularly emphasized. Nowadays our networks are getting more and more sensible to latency due to the proliferation of applications and services like VoIP, IPTV or online gaming where a low delay is essential for a proper performance and a good user experience.

Most of this unnecessary delay is created by the misbehaviour of many buffers that populate Internet. Instead of performing the task for what they were created for, absorbing eventual packet bursts to prevent loss, they deceive the sender's congestion control mechanisms into believing that the current path to the destination has more bandwidth than it really has. When the loss event occurs, if it does, it's too late and the damage on the path, in terms of additional transmission time, has been done.

On this bachelor thesis we will try to throw light over an specific solution that aims to reduce the extra delay produced by these bloated buffers: Active Queue Management. We have tested a bunch of AQM algorithms with different TCP modifications in order to understand the interactions between these two mechanisms. We performed simulations testing various characteristic scenarios like Transoceanic links or Access link scenarios, among other.

Resumen

En los últimos años se ha ido poniendo énfasis particularmente en la importancia del retraso sobre la capacidad. Hoy en día, nuestras redes se están volviendo más y más sensibles a la latencia debido a la proliferación de aplicaciones y servicios como el VoIP, la IPTV o el juego online donde un retardo bajo es esencial para un desempeño adecuado y una buena experiencia de usuario.

La mayor parte de este retraso innecesario se debe al mal funcionamiento de algunos búferes que pueblan internet. En vez de desempeñar la tarea para la que fueron creados, absorber eventuales ráfagas de paquetes con el fin de prevenir su pérdida, hacen creer al mecanismo de control de congestión que la ruta hacia el destino actual tiene más ancho de banda que el que posee realmente. Cuando la perdida de paquetes ocurre, si es que lo hace, es demasiado tarde y el daño en el enlace, en forma de tiempo de transmisión adicional, ya se ha producido.

En este trabajo de fin de grado intentaremos arrojar luz sobre una solución específica cuyo objetivo es el de reducir el retardo extra producido por esos hinchados búferes, la Gestión Avanzada de Colas o Active Queue Management (AQM). Hemos testeado un grupo de estos algoritmos AQM junto con diferentes modificaciones del control de congestión de TCP con el fin de entender las interacciones generadas entre esos dos mecanismos, realizando simulaciones en varios escenarios caracterísiticos tales como enlaces transoceánicos o enlaces de acceso a red, entre otros.

Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. Goals . . . 1
- 1.3. Social and Economic Framework . . . 2
- 1.4. Legislative and Regulatory Framework . . . 3
- 1.5. Thesis Structure . . . 3

2. State of the art **5**
- 2.1. TCP Congestion Avoidance Algorithms . . . 5
  - 2.1.1. TCP Tahoe . . . 5
  - 2.1.2. TCP Reno . . . 7
  - 2.1.3. TCP NewReno . . . 7
  - 2.1.4. TCP Cubic . . . 9
  - 2.1.5. Compound TCP . . . 10
  - 2.1.6. LEDBAT . . . 12
  - 2.1.7. BBR . . . 14
- 2.2. Queue management inside routers . . . 15
  - 2.2.1. Random Early Detection . . . 16
  - 2.2.2. Random Exponential Marking . . . 17
  - 2.2.3. Adaptive Random Early Detection . . . 17
  - 2.2.4. CoDel . . . 18
  - 2.2.5. Fair Queuing CoDel . . . 19
  - 2.2.6. CoDel-DT . . . 20
  - 2.2.7. PIE . . . 20
- 2.3. Related work . . . 21

3. Experimental Setup **25**

4. Access Link **29**
- 4.1. New Reno . . . 29
- 4.2. Compound TCP . . . 30
- 4.3. Cubic . . . 33
- 4.4. LedBat . . . 34

5. Datacenter Scenario **37**
- 5.1. NewReno . . . 38
- 5.2. Compound TCP . . . 38
- 5.3. Cubic . . . 40

5.4. LedBat 42

6. Satellite Scenario **45**
6.1. NewReno 46
6.2. Compound TCP 48
6.3. Cubic 49
6.4. LedBat 49

7. Transoceanic Scenario **53**
7.1. NewReno 54
7.2. Compound TCP 55
7.3. Cubic 56
7.4. LedBat 57

8. Conclusions **61**
8.1. Conclusions 61
8.2. Future work 63

A. Plan and Budget **65**
A.1. Plan 65
A.2. Budget 66

References **69**

Acronyms **75**

List of Tables

2.1. Related work 24

3.1. AQM Simulation parameters 26

List of Figures

2.1. New Reno cwnd evolution 9
2.2. CUBIC window progression 11
2.3. CTCP window progression 13

3.1. Topology of the experiment 25

4.1. Topology of the Dumbbell Scenario 29
4.2. NewReno: Delay vs. Interval 30
4.3. NewReno: Loss vs. Interval 30
4.4. NewReno: Load vs. Interval 31
4.5. CTCP: Delay vs. Interval 31
4.6. CTCP: Loss vs. Interval 32
4.7. CTCP: Load vs. Interval 32
4.8. CUBIC: Delay vs. Interval 33
4.9. CUBIC: Loss vs. Interval 33
4.10. CUBIC: Load vs. Interval 33
4.11. LEDBAT: Delay vs. Interval 35
4.12. LEDBAT: Delay vs. Interval 35
4.13. LEDBAT: Load vs. Interval 35

5.1. Topology of the Datacenter Scenario Scenario 37
5.2. NewReno: Delay vs. Interval 38
5.3. New Reno: Queue vs. Interval 39
5.4. New Reno: Delay vs. Interval 39
5.5. CTCP: Delay vs. Interval 39
5.6. CTCP: Loss vs. Interval 40
5.7. CTCP: Load vs. Interval 40
5.8. Cubic: Delay vs. Interval 41
5.9. Cubic: Loss vs. Interval 41
5.10. Cubic: Load vs. Interval 42
5.11. Ledbat: Delay vs. Interval 42
5.12. Ledbat: Loss vs. Interval 43
5.13. Ledbat: Load vs. Interval 43

6.1. Topology of the Satellite Scenario 45
6.2. Cummulative distribution of delay on Satellite scenario 46
6.3. NewReno: Delay vs. Interval 47
6.4. NewReno: Loss vs. Interval 47

6.5. NewReno: Load vs. Interval 48
6.6. CTCP: Delay vs. Interval 48
6.7. CTCP: Loss vs. Interval 49
6.8. CTCP: Load vs. Interval 49
6.9. Cubic: Delay vs. Interval 50
6.10. Cubic: Loss vs. Interval 50
6.11. Ledbat: Delay vs. Interval 50
6.12. Ledbat: Loss vs. Interval 51
6.13. Ledbat: Load vs. Interval 51

7.1. Topology of the Transoceanic Scenario 53
7.2. fq-Codel: Delay vs. Simulation time 54
7.3. NewReno: Delay vs. Interval 55
7.4. NewReno: Loss vs. Interval 55
7.5. NewReno: Load vs. Interval 55
7.6. CTCP: Delay vs. Interval 56
7.7. CTCP: Loss vs. Interval 56
7.8. CTCP: Load vs. Interval 57
7.9. Cubic: Delay vs. Interval 57
7.10. Cubic: Loss vs. Interval 58
7.11. Cubic: Load vs. Interval 58
7.12. Ledbat: Delay vs. Interval 58
7.13. Ledbat: Loss vs. Interval 59
7.14. Ledbat: Load vs. Interval 59

Chapter 1

Introduction

Since the beginning of Internet, engineers and scientists have been worried about network congestion. Some of these concerns caused the creation and implementation of several congestion control techniques that have survived until our days in the shape of TCP Congestion avoidance algorithms.

Most of these algorithms make use of dropped packets on a connection to sense congestion and act consequently. With the addition of buffers all across the network, the correct behaviour of this type of congestion notification has been severely damaged. Such added buffers prevent packet losses, do not letting the transmitter to know about bottlenecks on the connection and thus, adding unnecessary delay to the network path by systematically filling the buffers. This problem has been getting worse as time goes by.

1.1. Motivation

Nowadays network has changed, delay is slowly replacing throughput as the key factor on a modern network and many providers have already changed their infrastructures to support more advanced queue management mechanisms like Random Early Detection (RED), Proportional Integral controller Enhanced (PIE) or Controlled Delay (CoDel) (which will be reviewed on section 2.2) in an attempt to reduce the delay that the loss based congestion control algorithms induce on the network.

These queue management algorithms drop packets to warn the sender about congestion, replacing the legacy TCP congestion notification and triggering the congestion avoidance mechanisms on the sender size. But not all the TCP flavours use the same techniques to detect congestion. How these new methods interfere with built-in congestion-avoidance algorithms included on TCP?

1.2. Goals

The purpose of this thesis is to simulate different scenarios and discuss the interactions between the major congestion avoidance algorithms in use nowadays

and the queue management strategies proposed in the last years to mitigate bufferbloat [28], which is where the buffers of a router are always full, adding unnecessary extra delay to the network and failing at its goal of absorbing eventual packet bursts.

In order to do so, we added several AQM algorithms to the TCP test suite created by the Internet Congestion Control Research Group (ICCRG) [6] and analyzed the interactions produced between these algorithms and the main TCP flavors used nowadays on the network. We hope to throw some light over the real interactions that occur on the network core and figure out if the benefits provided with these mechanisms are consistent and viable.

1.3. Social and Economic Framework

As we stated previously, delay accumulated on our communication networks plays an unnoticeable but important role in our society nowadays. The effects of such delays goes from video-calls and another interactive applications [18] with poor user experience to computer games [7] [78] [14] [16] suffering the effects of the so-called "lag", without forgetting the millions and millions of economical losses if we stare at the new cutting edge High Frequency Trading systems, in which every nanosecond costs money [40]. We cannot deny that the failure of the so called thin clients was partly caused by the heavy delays the users had to withstand when using one of these machines [72].

An illustrative example about how critical these delays could become on a future not so distant are the teleoperation of surgical robots [47], in which every millisecond is important and could mean the difference between life and death. According to [63], the future applications for this type of technology could be key on some scenarios, for instance, reducing the exposure of real human surgeons on the battlefield.

The health of those industries is key on our modern economies. According to SuperData research firm [61], games generated a revenue of 91 billion dollars on 2016, being most of this money generated by games with a heavy use of online technologies, and these numbers do not seem to slow down for the next year. The same happens on the phone industry when reports indicate that companies are moving to a total conversion from conventional PSTN networks to VoIP technologies [17]. We could also mention the VoIP features that instant messaging companies are adding to their apps, and the impact that these features could have globally, let's remember that Whatsapp (one of the most important instant messaging companies) counts with 900 millions active users and its still growing.

An example of how latency is crucial on delivering multimedia transfer are the importance that Youtube or Netflix gives to that aspect on the design of their infrastructure as it is stated on [24] and [2].

The network is changing faster and faster from a model in which the heavy

weight was done locally and the Internet was just a channel to a model in which the real work is done on the flight and we want to see the changes reflected with the same speed as before but worldwide instead of locally. The increasing importance of delay on our communication networks will rise over time due to the necessities imposed by the new cloud-based services that are appearing now. Delay is crucial on most of these new technologies that are flourishing.

1.4. Legislative and Regulatory Framework

The legislation on this topic is nonexistent. The main reason is that the details we are discussing om this thesis belongs to highly technical aspects of the telecommunications that are not subject of legislation. Nevertheless, latency should have been included on the IET/1090/2014 [50] law which regulates the quality of service parameters that the operators need to cover. This could have forced the Spanish's telecommunication operators to reduce the delay experienced on their infrastructure.

Although the adherence to its standards is voluntary, the Internet Engineering Task Force (IETF) is the most important organization that works towards a regulatory framework on TCP/IP networks. Since its creation it has encouraged the work on several engineering aspects related to this thesis. It deployed a dedicated team called Internet Congestion Control Research Group (ICCRG) to address the challenge of optimizing TCP Congestion Control. Since its creation this team had an influence on the design of new TCP congestion control algorithms like CTCP or Cubic.

Besides that, other group created by the IETF called Reduce Internet Transport Latency (RITE) works towards the standardization of methods to measure, and control the delay on TCP/IP based networks.

1.5. Thesis Structure

This document is divided into eight sections in which we cover the full aspects for the development of the following thesis.

On this first chapter, we have written an introduction in which we explain briefly what took us to make this thesis. It also covers the social and economical aspects of our work and it also explains a little bit the legal and economical framework of this topic.

On the second chapter we will talk about the state of the art, dedicating the first section to explain the most important TCP congestion avoidance mechanisms that are in use nowadays. We will talk about the main aspects and differences of each one of them. The next section of this chapter is similar to the aforementioned, but in this case we will speak about some of the most extended Active Queue Management schemes that are in use and that we will evaluate, the mechanisms

that are inside its control logic, its flaws and strengths. Later on, we will take a look at the works previously published about this topic, highlighting its similarities and differences with this thesis and showing its conclusions.

The third chapter is devoted to explain our simulation setup, the tools we used and the topology we utilize to test it. This chapter serves as an introduction for the next four chapters in which we will show the results of the different scenarios we simulated: Access Link, Transoceanic, Datacenter and Satellite scenario. On each one of these four chapters, we will explain the characteristic features that makes it interesting to analyze, the parameters chosen to make our topology similar to these scenarios and we will comment the various results we obtained during those experiences.

The eighth section is a concise text explaining our conclusions and the reasons behind such that conclusions. We will also propose some improvements of out work as well as new research approaches. The end of this document will be filled with the budget of this projects as well as with the bibliography and a list of utilized acronyms.

Chapter 2

State of the art

2.1. TCP Congestion Avoidance Algorithms

This section is intended to act as a brief review about the most important TCP congestion control variations that are in use nowadays. For that reason we have chosen some of the most representative TCP modifications like CUBIC, which is the standard TCP extension used on Linux machines, Compound TCP which is implemented by default on Windows Operative Systems, LEDBAT which is one of the best representations of a TCP Congestion Control algorithm based on Low Priority Congestion Control and New Reno which is the basic scheme from where all these other TCP modifications came from.

These concepts will be useful in order to analyze the interaction between TCP congestion avoidance algorithms and the different queue management algorithms deployed on this experiment.

2.1.1. TCP Tahoe

TCP Tahoe is one of the first congestion control algorithms and the base in which the newest proposed versions like Reno or New Reno were based on. It was proposed by Van Jacobson and Mike Karels on 1988 [38] and implemented on BSD operative system, which codename was "Tahoe", in the same year. Tahoe was based on the principle of 'conservation of packets'. According to this principle, there should be an equilibrium point in which a link could work stable and keep that state by only adding a packet to the link just if another packet has left the link.

So, in order to obtain the best performance on a link, we would need two algorithms to address these two different phases. The first one would look for the equilibrium point and the second one would be in charge of keeping the connection near to that point. We would call these two phases Slow-Start and Congestion Avoidance, respectively. As we will see, such concepts will be utilized on the congestion control mechanisms that followed this first attempt.

These two algorithms try to control the amount of data sent to the link so as to

achieve the maximum available throughput. In order to do that, two parameters are used: The first one is the congestion window size, *cwnd*, which controls the number of bytes that can be sent in a row and the second one is *sstresh* which indicates the number of packets that can be sent until the connection enters into congestion-avoidance mode.

On Slow-start phase, while *cwnd* is less than *sstresh*, for every ACK received, the sender increases the window size by one packet. That is the so called Slow Start algorithm which is not so slow, in fact, it leads to an exponential window increase that aims to achieve the maximum possible bandwidth in the minimum possible time. When a loss happens, this *cwnd* is resetted to one packet.

When *sstresh* value has been reached, connection goes into congestion-avoidance mode and it starts to increase the window linearly, by adding $\frac{1}{cwin}$ to the actual window value. If some packet is lost, the sender resets the window to its initial value and resets the threshold to half the value of the congestion window when the loss happened. For instance, if we started the congestion-avoidance phase sending ten packets, and the loss occurred when the congestion window was thirty six, the threshold value would be eighteen and the congestion window will start again at ten packets.

TCP Tahoe also made use of another two different strategies to detect losses: Retransmission Timeout (RTO) estimation and Fast Retransmit. RTO estimation is intended to minimize the reaction time to a packet loss by adjusting the timeout to the network conditions and Fast Retransmit algorithm gives the sender the capability to repeat the transmission of a loss packet without waiting for its RTO timer to expire.

In order to estimate RTO, TCP Tahoe make use of three variables *srtt* (smoothed round-trip time), *rttvar* (round-trip time variation) and the proper RTO. It is initially set to 3 seconds, but when the first RTT measurement is obtained, the values are updated as follows:

$$\begin{aligned} srtt &= rtt \\ rttvar &= \tfrac{rtt}{2} \\ rto &= srtt + 4 \cdot rttvar \end{aligned} \tag{2.1}$$

Each time a new RTT measurement is adquired, the three values are recalculated as follows:

$$\begin{aligned} srtt &= (1-\alpha) \cdot srtt + \alpha \cdot rtt \\ rttvar &= (1-\beta) \cdot srtt + \beta \cdot (srtt - rtt) \\ rto &= srtt + 4 \cdot rttvar \end{aligned} \tag{2.2}$$

As you can see on 2.2, the smoothed RTT is a low-pass filter as [58] suggested with filter gains being α and β. Jacobson proposed [38] for such values 0.9 and 2 accordingly.

Although Retransmission timeout is able to detect every loss, its reaction time

is bounded by the RTT and the RTO which sometimes could not be fast enough. In order to address this problem, a fast retransmission mechanism called Fast Retransmit was implemented on TCP Tahoe.

So, if some packet is lost, the receiver keeps sending the acknowledgment packet that corresponds to the last packet received before the packet loss until the missing packet is received. As a result of this, the sender will receive more than one ACK with the same sequence number. If it receives three duplicated ACK's, it infers that this particular packet has been lost and sends it again without waiting for the retransmission timer to expire. TCP Tahoe behaves on this event like on a common loss event, by resetting the Congestion Window and starting again the Slow-Start and Congestion Avoidance phases.

2.1.2. TCP Reno

TCP Reno is similar to TCP Tahoe but with the addition of another congestion control mechanism called Fast Recovery [70]. This mechanism comes from differentiating between minor losses caused by little congestion peaks or eventual transmission problems (at the physical layer, for instance) and major losses caused by real link congestion. By making such differentiation, it could be possible to address these minor losses not so aggressively, thus conserving the capacity before the loss event on some cases.

So, understanding the reception of duplicated acknowledgements as a minor loss event that does not implies necessarily congestion (cause other packets have been able to get to the receiver), Jacobson proposed an alternative way of handling such losses called Fast Recovery.

By using this modification, when an event with suchlike characteristics have place, the sender recalculates its threshold value just like in Congestion-Avoidance mode but, instead of setting the congestion window back to the original window size, it sets the window to $sstresh + 3$, skipping the Slow Start phase and going directly to Congestion Avoidance phase.

2.1.3. TCP NewReno

TCP NewReno [32] is an evolution of the original TCP Reno explained on section 2.1.2 which aims to keep window transmission full when it is in recovery mode by improving the Fast Recovery algorithm it was implemented on TCP Reno.

It addresses a specific drawback suffered by TCP Reno [21] that occurs when several concatenated loss events take place. On this situation, when TCP Reno receives an acknowledgement of some of the lost packets, it goes out of Fast Recovery phase thus not automatically retransmitting the remaining lost packets as it should be done on Fast Retransmit, waiting for their timers to expire instead.

Rather than doing that, New Reno tracks the acknowledgements received and

does not go out Fast Retransmit phase till all the lost packets have been acked.

As we stated before, the main variable that TCP New Reno and most of the modern TCP congestion management algorithms uses to take control of the congestion on a link is the window size [48], that is, the number of packets we can send in a row. By modifying this parameter, we are able to modify the traffic sent to a link and thus, reduce or increase the congestion on such the link. The optimal window size (if there is just one TCP flow on the link) would be equal to the Bandwidth Delay Product (BDP). As we do not know a priori the bandwidth of the link, the goal in the case of TCP New Reno and all the other loss-based congestion control algorithms is to take the congestion window to the highest value that would allow the link to work without suffering losses.

TCP New Reno, just like their predecessors, is composed of four algorithms: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery. It could also be divided into two main phases: Slow Start, where the algorithm with the same name takes place and Congestion Avoidance, when the congestion window is adjusted according to the Congestion Avoidance directives. On this phase, the Fast Retransmit and Fast Recovery algorithms could also work if some loss is produced.

Slow Start and Congestion Avoidance algorithms on New Reno follow the same principles as the older Tahoe and Reno implementations, just changing a few details like the default window size which is usually initialized to three times the Maximum Segment Size, or *sstresh* which is 64 Kb by default.

Slow-start phase is particularly important on some applications like web browsing, when sometimes the transmission of data is so short that TCP is not even able to leave Slow-start phase behind. Fortunately, this problem has been minimized with some improvements on the application layer, regarding the way HTTP make the connection and download the assets, but those details are out of the scope of this thesis.

Another way to maximize the number of information transmitted on this phase could be to increase the initial congestion window from the standard 3MSS to a higher value like 10MSS [15]. It has been proven [20] that it could have a moderate benefit for short lived connections although it is obvious that a larger initial window increases the burstiness of the traffic and this could affect significantly to the average queuing delay under some circumstances.

Some other reports [27] indicate that the impact of the beforementioned modification on the latency is huge enough to advice not to make such that modification. Nevertheless, this report is targeted to a very specific service (web browsing) which has its own idiosyncrasies and its conclusions are based on an old protocol version (HTTP/1.1). Besides that, this draft talks about queues presumably without any active queue management policy. We will try to figure out if these type of managers could mitigate the latency problem and, at the same

time, improve overall throughput when using this increased initial window.

```
win = 3
sstresh = 64Kb
if(ack){
   if(win < sstresh) win = win + 1
   if(win > sstresh) win = win + (1/win)
}
if(packet_loss) {
        sstresh = win/2;
        win = 1
if(three_ack){
        sstresh = win/2
   win= sstresh + 3
}
```

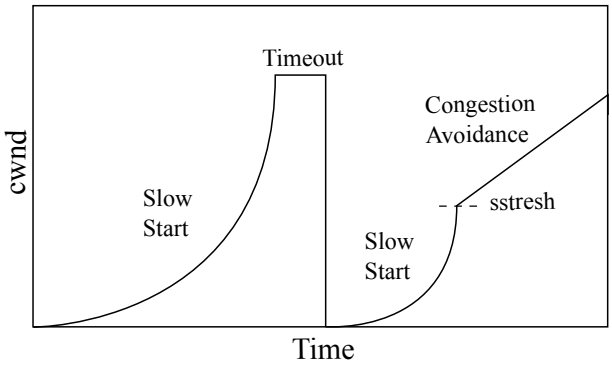


Figure 2.1: New Reno cwnd evolution

2.1.4. TCP Cubic

Cubic [31] is the current default TCP algorithm implemented on Linux machines since kernel 2.6.1. It is an improved version of BIC-TCP that tries to simplify the window control and enhance it's friendliness with standard TCP.

It is supposed to work better than TCP New Reno on Long Fat Networks (LFN), networks with high bandwidth and high delay because it minimizes the time needed to reach the full capacity on links with very large Bandwidth Delay Product (BDP). It does this by replacing the linear window growth function with a cubic function in order to increase link utilization.

Cubic has two phases. In the first one, it rapidly increases towards the window size it had just before the last congestion event happened. We will call this threshold, w_{max}. As it reaches this value, the growth of the congestion window is slowed down, behaving like a concave function. The initial fast window size rising provides a shorter recovering time after a loss event compared to New Reno while the flattening that occurs as congestion window approaches to w_{max} helps to

minimize the losses in case the congestion window has truly reached the network limit.

When w_{max} is surpassed, it goes into the second phase when the window increase function become convex, trying to probe the network for the highest possible congestion window in an aggressive manner. The objective of this phase is to rapidly find the maximum possible transmission window the link is able to withstand.

When a loss happens, w_{max} value is updated by using the formula 2.3, being β a constant multiplication decrease factor which default value on TCP Cubic is 0.2.

$$w_{max}(t) = (1 - \beta) \cdot w_{max}(t - 1) \tag{2.3}$$

On 2.4 we can observe the cubic polynomial formula that controls the congestion window value update when an acknowledgement is received and draws the representative cubic shape that gives name to this TCP algorithm.

$$w(t) = C(t - K)^3 + w_{max} \tag{2.4}$$

As we can see, this function is just a customized version of the cubic function x^3 where C is a scaling factor which default value is 0.4, t is the elapsed time since the last window reduction, w_{max} is the recalculated window size just before the last loss event and K is a factor that is updated at the time of the last congestion event by using the formula 2.5.

$$K = \sqrt[3]{\frac{w_{max} \cdot \beta}{C}} \tag{2.5}$$

In order to increase fairness, TCP Cubic includes a modification inherited from TCP BIC called *Fast Convergence* [62]. This modification is designed to recalculate the congestion window when the available capacity on the channel decreases. In order to do that, the last congestion window before a loss event is remembered. We will call this value w_{last}. When a new loss occurs, this value is compared to the actual w_{max}. If $w_{max} < w_{last}$ it means that the capacity of the link has been reduced. Instead of recalculating w_{max} as we did before, the formula 2.6 is utilized to obtain a w_{max} value slightly smaller than usual.

$$w_{max}(t) = \frac{2 - \beta}{2} \cdot w_{last} \tag{2.6}$$

On figure 2.2 we can observe an example of how the different mechanisms mentioned above influence the congestion window.

2.1.5. Compound TCP

Compound TCP (CTCP) [71] is an algorithm developed by Microsoft that was introduced on 2008 as part of Windows Vista. It basically adds a delay-based

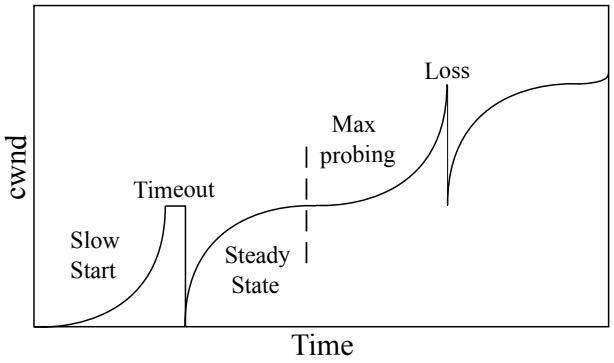


Figure 2.2: CUBIC window progression

component to TCP New Reno with the purpose of improving link utilization on high-speed and long distance networks as they stated on their published paper. Like Cubic, it tries to fully utilize the capacity of the link by converging to the maximum possible window size faster than TCP New Reno but, at the same time, it tries to be fair with the other competing TCP flows in the link.

As a result of this, Compound TCP is an hybrid between a pure delay-based congestion control (based on HS-TCP) and a classic loss-based congestion control. The included delay-based component helps to increase aggressively the window when the network is under-utilized and reduce the sending rate when it senses that the link is fully utilized, that is, in the presence of queuing delay.

Compound TCP algorithm is similar to TCP New Reno but, instead of having one congestion window variable, it maintains two independent windows: the loss-based window *cwnd* and the delay based window *dwnd*. If the algorithm is in congestion-avoidance phase, the actual window is the minimum between the sum of these two windows and the advertised window of the receiver as it is shown on equation 2.7.

$$wnd = min(cwnd + dwnd, awnd) \tag{2.7}$$

It is important to highlight that *dwnd* will not be taken into account at the initial Slow Start phase of any TCP transmission cause the designers thought the exponential slope on this phase is fast enough to provide a proper transmission beginning. On congestion avoidance phase, as *cwnd* works exactly like in TCP New Reno, the window size will be increased on every acknowledgement received by following the New Reno formula, but using the window calculation formula 2.7 as it is written on formula 2.8.

$$wnd = wnd_{last} + \frac{1}{wnd_{last}} \tag{2.8}$$

Now let us explain how this *dwnd* value is chosen. The delay component uses a variable RTT_{base} as transmission delay estimator. With this measure we can figure out the throughput we expect to receive as on equation 2.9, and if we compare this

estimator with the actual throughput (equation 2.10) it is possible to estimate the number of packets that are stuck in the bottleneck as in the formula 2.11.

$$thr_{expct} = win/RTT_{base} \tag{2.9}$$

$$thr_{now} = win/RTT \tag{2.10}$$

$$diff = (thr_{expct} - thr_{now}) \cdot RTT_{base} \tag{2.11}$$

If this value is bigger than a predefined target value (γ), then it is assumed that the link is congested and the value of *dwnd* shoud decrease. If the value of the difference is smaller than the threshold value otherwise, it is determined that the link is underutilized and the delay-based congestion window should be increased. Threshold value γ is heuristically assigned being 30 packetes the default value suggested by the creators of CTCP.

Until now, we just saw how the delay-based component of the CTCP protocol detects congestion. The general outline then for the congestion window control function is the one depicted on equation 2.12.

$$wnd(t+1) = \begin{cases} wnd(t) + \alpha \cdot wnd(t)^k & no\ loss \\ wnd(t) \cdot (1-\beta) & loss \end{cases} \tag{2.12}$$

By removing the loss-based component, the function could be particularized for the delay-based window as follows on equation 2.13.

$$dwnd(t+1) = \begin{cases} dwnd(t) + \alpha \cdot xwnd(t)^k - 1 & diff < \gamma \\ dwnd(t) - \zeta \cdot diff & diff \geq \gamma \\ dwnd(t) \cdot (1-\beta) - \frac{cwnd}{2} & packet\ losses \end{cases} \tag{2.13}$$

The previous formula 2.13 is created by taking into account that the delay-based component acts as a complement of the New Reno algorithm loss-based algorithm. For instance, in the increase phase, when target γ is bigger than *diff*, increment of *dwnd* will be conditioned by the loss-based component which will increase by one packet. When there is a loss, it also does the same regarding the reduction provided by *cwnd*. In such that situation, ζ is a very important coefficient which controls the speed on the reduction of *dwnd* when there is congestion. An example of how the congestion window evolves on Compound TCP can be seen on figure 2.3.

2.1.6. LEDBAT

Low Extra Delay Background Transport (LEDBAT) [68] is a pure delay-based congestion control algorithm designed to use just the spared bandwidth on an end-to-end link while trying to keep queuing delay as low as possible.

In order to achieve this, it measures the one-way delay and reduces its rate when delay increases. This behaviour make LEDBAT less aggressive than standard New

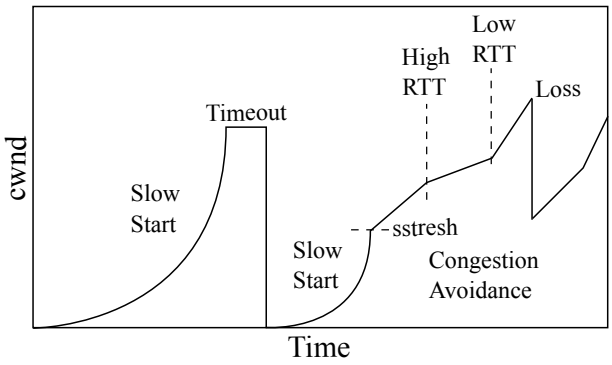


Figure 2.3: CTCP window progression

Reno TCP, reacting faster to congestion events and not inducing extra queuing delay in the network.

LEDBAT senses congestion on a way very similar to CTCP does, but instead of using Round Trip Time, it employs one-way delay. This parameter has the advantage of being more isolated with respect to other traffic flows sharing the same link, but it requires collaboration between the sender and receiver which has added timestamps to the packets sent.

Then, by calculating the difference between the base delay and the current delay experienced, we are able to find out an estimator of the delay present on the queue as in equation 2.14. The base delay is defined as the minimum delay present on the link and is calculated like it is shown on equation 2.15.

$$del_{queue} = del_{now} - del_{base} \tag{2.14}$$

$$del_{base} = min(del_{base}, del_{now}) \tag{2.15}$$

The final value utilized on the congestion window control function is the target offset depicted on figure 2.16 which represents the normalized difference between the measured queuing delay del_{queue} and the specified target delay q_{target}, which is 25 ms by default.

$$off_{target} = \frac{d_{target} - d_{queue}}{d_{target}} \tag{2.16}$$

With this value and another configurable parameter that is the gain, defined as the rate at which the congestion window *cwnd* responds to the changes in queuing delay, the congestion window control functions is configured as it is explained on equation 2.17.

$$cwnd(t) = cwnd(t-1) + \frac{gain \cdot off_{target}}{cwnd(t-1)} \cdot bytes_{acked} \cdot MSS \tag{2.17}$$

As it is stated on [65] this function established a direct relation between the

target delay and the window size, aggressively increasing the window when the current delay is well below the target delay and slowing down such tendency when it draws closer to the desired value.

If a packet loss is detected, the congestion window is reduced by half its value, just like in TCP New Reno, so in the worst scenario LEDBAT would act as a "standard" TCP congestion control.

It is very interesting to test this algorithm together with different AQM mechanisms because Low Priority Congestion Control (LPCC) strategies such that the methods deployed on LEDBAT have been one of the alternatives to AQM proposed by the research community in order to reduce queuing delay. Besides that, LEDBAT has been widely deployed, showing a proper performance [64] in relation with its intended goals. Apple uses this protocol to distribute software updates and uTorrent [65] added it to its uTP protocol.

2.1.7. BBR

Bottleneck Bandwidth and RTT (BBR) saw the light on 2016 when an article written by Neal Cardwell, Van Jacobson et al. was published on ACM journal [9]. On this paper, the group of researchers presented a new TCP algorithm which seemed to be able to adapt itself to the bottleneck link on the connection and get closer to the link's operating point than any other algorithm before.

This algorithm make use of a technique called TCP Pacing [4] which consists on distribute the data we need to send over a period of time (usually a RTT) to reduce burstiness. This detail is essential in order to make BBR work as it is intended to work because the main parameter it uses to modify transport flow behaviour is the pacing rate.

BBR obtains an approximation of the link's optimal operating point by estimating two parameters: Round Trip Time, $RTProp$, and Bottleneck bandwidth, $BtlBw$. Then it tries to match the pacing rate to the bottleneck bandwidth in order to not to generate unnecessary queuing delay [10]. So to accomplish that, pacing rate is defined as the bottleneck bandwidth, $BtlBw$ by a variable pacing gain that would be modified depending on the current phase of the algorithm and the status of the link.

In order to estimate $BtlBw$, BBR maintains a windowed max of 10 RTT of length, which is updated on every acknowledgement received with the delivery rate (detailed information about this estimation can be found at [13]). To find out $RTProp$ it does something similar, by utilizying a min filter of ten seconds of length which is updated on every ack received with the RTT.

BBR goes through several phases over the connection in order to achieve that. When the transmission is started, BBR goes into Startup phase. During this phase, BBR tries to find out the bottleneck bandwidth by doubling the sending rate on

each round. This is done by setting the pacing gain to $2/ln(2)$. When the algorithm detects it has reached the limit (by looking to differences between the sending rate and the delivery rate), it falls into drain phase in which it tries to drain the extra queue created on the bottleneck buffer during the earlier phase, by setting the pacing rate to $ln(2)/2$. When data in flight is equal to the estimated BBR, this phase ends and the algorithm begins oscillating between probe bandwidth phase ($ProbeBW$) and probe RTT phase ($ProbeRTT$).

When it is on $ProbeBW$ phase, it probes for more bandwidth by slightly modifying the pacing gain on a cycle of six gain values, each one of them during a Round Trip Time. Gains that compose the cycle are 1.25, 0.75, 1, 1, 1, 1. This values are chosen in order to avoid adding extra queue to the bottleneck link, if the connection is already working at its maximum available bandwidth. The initial phase inside the gain cycle is randomized with the aim of avoiding flow synchronization and encourage fairness between flows sharing the link. This phase lasts 10 seconds and when it is over, it spends 200ms on the $ProbeRTT$ phase, trying to obtain the closest possible value to the real RTT on the network path. To do so, congestion window is reduced to a minimum packet of four MSS. Reducing the inflight data to that amount helps draining eventual queues along the link and obtain a RTT estimation closer to the real value.

Besides the positive results shown on the paper made by its creators, some other early evaluations [33] cast doubts upon the real impact of this new congestion control algorithm and its capacity to fight bufferbloat problem. Such report warns about concerning problems like an increased overload on the bottleneck link or an important unfairness to loss based congestion control flows which leads to heavy losses and big buffering.

2.2. Queue management inside routers

In this section we are going to explain the Active Queue Management (AQM) algorithms that we are going to evaluate on this article. Such that algorithms are designed to replace simple FIFO Tail Drop algorithm as queue manager inside routers.

Tail Drop simply drops the new packets that arrives to a full queue until the queue is empty enough to accept new packets, as its name suggests, it drops the tail of the queue. The multiple drawbacks that Tail Drop has might have escalated with the evolution of our networks towards interconnected links with high BDP [12], resulting in an unacceptable loss of bandwidth, a delay rise and a huge queue size oscillations also called jitter.

On the other hand, AQM provides a proactive queue size control which is aimed to remove persistent queues, reduce queue size (thus reducing queuing delay), provide fairness between competing TCP flows and protect TCP self-clocking [1].

AQM has two main goals. The first one is to sanitize buffers by removing

persistent queues (also known as bad queues) while allowing the buffer to store occasional bursts of packets without damaging the throughput of the link. Bad queues are those queues that maintain a stable level of packages always enqueued, inducing a persistent and unnecessary queuing delay which is very difficult to remove by the ends of a connection because it is virtually invisible to them.

The second one is to allow proper operation of TCP congestion control, preventing buffers to interfere with congestion detection by absorbing crucial packet losses which have the purpose of warning the sender about congestion on the link.

In order to deeply understand these algorithms we have to take a look to their inner mechanisms. Each one of these managers has three main components. The first one, called Congestion Indicator is any kind of metric that models congestion on the router queue. For instance, values like queue size, average RTT, packet sojourn time or specific buffer events could work as congestion indicators.

The second component is the control function, the one that manages the way the packets are signaled. It is triggered and controlled by the congestion indicator. The third component is the feedback mechanism, that is, the type of signal delivered by the control function in order to generate a reaction on the server side. Such that feedback mechanism could consist on packet dropping or packet marking by using Explicit Congestion Notification (ECN).

2.2.1. Random Early Detection

Random Early Detection (RED) and its derivatives were the earlier attempts to improve queue management [26]. The original Random Early Detection algorithm took an Exponential Weighted Medium Average (EWMA) of the queue size q as congestion indicator, that we will denominate here q_avg. This value is updated on every packet arrival. The election of a EWMA to calculate the congestion indicator helps to obtain a smoother value thus removing small non critical queue length changes [75], like bursts, but it also makes RED slower when there is sudden packet arrival rate risings.

$$\bar{q}(t) = (1 - w_q)\bar{q}(t) + w_q q(t+1) \tag{2.18}$$

Under a predefined threshold q_{min}, the algorithm is deactivated but when queue size surpasses that threshold, control function starts dropping packets with a linear probability $p(q_{avg})$ until queue size decreases or until it goes above q_{max} when the dropping probability becomes 1. This behavior is exemplified on equation 2.19. Notice that p_{max} is the dropping probability when $q_{avg} = q_{max}$, so p_{max} has to be always lower than 1.

$$p(q_{avg}) = \begin{cases} 0 & q_{avg} < q_{min} \\ p_{max} \cdot \frac{q_{avg} - q_{min}}{q_{max} - q_{min}} & q_{min} \leq q_{avg} \leq q_{max} \\ 1 & q_{avg} > q_{max} \end{cases} \tag{2.19}$$

Although this algorithm was the first AQM existent and it meant an great advance compared to Tail Drop, it had several problems. First of all it was the difficult (even impossible) to tune heuristically the three configurable parameters $(p_{max}, q_{min}, q_{max})$ for the algorithm to be useful on multiple scenarios or under different network conditions.

The chosen congestion indicator has also been put into question. The length of the EWMA could make RED too sensitive against burst of packets [66] or increase its reaction time against real congestion events [22]. And, as has been amply demonstrated, queue length is definitely not a good congestion indicator [23]. The use of such that metric generates a strong deterioration on both throughput and delay when working at high traffic loads [35].

2.2.2. Random Exponential Marking

Random Exponential Marking (REM) [8] is a modification of RED which uses a different marking probability function as well as a different parameter to measure congestion on the link. Instead of using queue size, which has been pointed out as a bad predictor for congestion, REM uses as congestion indicator a variable called price $p(t)$.such that price is updated periodically based on rate $r(t)$ and queue $b(t)$ mismatch as we can see on equation 2.20. The price is incremented if the weighted sum of this two calculations is positive, sensing congestion, it is zero if the price is stable and it is negative if the congestion decreases.

$$p(t+1) = p(t) + \gamma \cdot (\alpha \cdot (q(t) - q_{ref}) + (r(t) - r_{ref})) \tag{2.20}$$

As it is difficult to know beforehand the reference rate for a link, sometimes the rate mismatch is replaced by the queue growth mismatch $(q(t) - q(t-1))$.

Control function then drops the packets arriving with a probability $m(t)$ as it is shown on equation 2.21, being ϕ a constant greater than zero that modifies the response of REM algorithm when it is on dropping mode, making it more or less aggressive.

$$m(t) = 1 - \phi^{-p(t)} \quad \phi > 1 \tag{2.21}$$

2.2.3. Adaptive Random Early Detection

After RED was released, a bunch of proposals [77] that tried to beat RED were developed. REM was one of these, but the list is endless: Gentle RED, Stabilized RED, Dynamic RED, Double Slope RED, Loss ratio based RED, Modified RED,BLUE, Yellow, GREEN, BLACK, DREAM, Fair RED, Balanced RED, Short-lived flow friendly RED, CHOKe...

But, as we previously stated when we explained RED, these first algorithms had lot of problems. One of the most important and recurrent problems was that all their parameters were hard to tune in order to have transverse performance

improvement across different scenarios and situations and some of them would require different values depending the actual status of the network (traffic load, actual throughput, number of flows, etc). That is the main reason why nor RED neither its modified versions was never broadly implemented across Internet or just deactivated by default in most of the devices. In order to fix these problems, a parameter-less version was developed called Adaptive Random Early Detection (ARED) [25].

Adaptive Random Early Detection dynamically changes p_{max} to maintain the average queue size q_{avg} as close as possible to a target value q_{target} that is at the half way point between min_{th} and max_{th}. ARED periodically compares q_{avg} with q_{target} and updates p_{max} in small steps by using an AIMD strategy as it is explained on equation 2.22. p_{max} is not allowed to exceed 0.5 or fall below 0.01, to limit its influence on the link.

$$p_{max} \leftarrow \begin{cases} p_{max} + \alpha & q_{avg} > q_{target} \\ p_{max} \cdot \beta & q_{avg} < q_{target} \end{cases} \tag{2.22}$$

It has been proved [43] that ARED behaves way better than RED on most situations, converging more quickly than RED and providing a predictable average queuing delay. It aso shows an improved stability independent of the number of flows on the link [44].

2.2.4. CoDel

Controlled Delay (CoDel) is a scheduling algorithm that uses packet-sojourn time through the queue as congestion indicator [52]. If queue is large enough and the minimum packet-sojourn time experienced over an specific interval of time surpasses a predefined threshold, then the control function is turned on and starts dropping packets every interval t. Such that interval is modified by using function 2.23.

$$t = \frac{t_{x-1}}{\sqrt{n_{dropped}}} \tag{2.23}$$

Both the threshold and the dropping interval have been chosen by taking into account the usual characteristics on Internet, trying to obtain a global setting that could improve bufferbloat problem on most of the possible scenarios. Nevertheless, some specific situations, like datacenter traffic or satellite links, could require some special tuning.

By default, t is set to 100 ms. As you can see, according to the control function such that interval decreases linearly when there is congestion until the sojourn time is below the threshold. We have changed this value in our experiments, thus making the algorithm less or more aggressive. It's suggested on [52] that the threshold, also known as "setpoint", should be a 5% of the interval t, that is, 5 ms. We also made experiments by changing such that threshold even though it is not intended to be modified.

The purpose of every AQM manager is to warn TCP about buffer congestion and control the delay by sending such that warnings to the ends of the connection. For that reason, in order to obtain an optimal operation Codel's interval should be large enough to give to TCP congestion control enough time to face losses produced by CoDel. The best value then, has to be larger than the link's round-trip time but without falling too far from that value because we do not want to store unnecessary packets on our buffer either. It has been proven that the default interval of 100 ms fits quite well across a wide range of scenarios with RTT's that goes from 10 milliseconds to 1 second [54].

On this brief explanation of CoDel we talked about the interval in which the algorithm checks for the minimum packet sojourn time and the interval used on the control loop phase to space the packet dropping without making any differences between them but in fact, these two values, although most of the times are the same, could have different values. Modifying the dropping interval could lead to a more aggressive way of performing but maintaining the reaction time of the algorithm.

2.2.5. Fair Queuing CoDel

Fair Queuing CoDel (fq-CoDel) [34] is a CoDel algorithm that tries to isolate each different flow on a network by classifying the incoming packets into different queues. fq-CoDel generates a hash with the IP protocol number, destination IP, destination port, source IP and source port in order to perform such that classification.

These isolated queues are managed by a different instance of CoDel and all of them are managed together with an slightly modified Deficient Round Robin (DRR) scheduler [69] that manages the amount of bytes that can be sent by each one of the isolated queues. This allows fq-CoDel behave like the original CoDel algorithm when it has only one flow. These flow differentiation solution offers obvious advantages and it is supposed to outperform CoDel at the expense of more processing power and complexity.

The DRR version that this algorithm uses makes distinction between queues that generate standing queue long enough to last various rounds and new flows which doesn't, and it processes the new queues before the old ones. If a queue is empty, it is deleted. This detail makes flows with acceptable transmission rates to be processed before flows that creates bufferbloat because they are treated like new ones. This first classification helps to remove congestion from critical flows like acknowledgements, DNS or HTTP requests. After this first classification, there is another regular round-robin scheme, as in the original DRR.

Another remarkable thing is that the quantum given to the differentiated flows on each round is measured on bytes and not on packets, to not to penalize flows regarding their packet size. The creators suggest that such value it is related with the link's bandwidth so values smaller than the standard MTU should suit better

on scenarios with limited bandwidth.

2.2.6. CoDel-DT

In spite of that CoDel has a solid theory base, it presents some problems when it has to be implemented onto real hardware. For instance, you have to process each packet that comes into your router in order to add it a time-stamp which means that every packed have to be enqueued and this detail could affect to the router's efficiency.

In order to address these details and obtain an easier hardware implementation, specially suitable for DOCSIS cable modems, an adapted version of CoDel called CoDel-DT was developed [74]. This modification uses delay prediction at enqueue time and tail drop instead of real time-stamping and head dropping. If this prediction is precise enough, CoDel-DT should show a performance as good as the real CoDel algorithm.

The latency estimator function developed by CableLabs is the same as the one utilized in the PIE algorithm.

2.2.7. PIE

Proportional Integral controller Enhanced (PIE) [57] is CoDel's main competitor. It has several characteristics that makes it more suitable for routers, just like CoDel-DT and it is parameterless.

It uses an estimation of the current queuing delay as congestion indicator instead of the real queuing delay, which relieves the router of the necessity of enqueue and mark with a timestamp all the packets. Such that estimator consists on ratio between the queue length, N, and the departure rate, μ like in equation 2.24.

$$\bar{q} = \frac{N}{\mu} \tag{2.24}$$

PIE uses the drop probability depicted on equation 2.25, that takes into account the target delay (q_{ref}) and the trend since the last update, by assigning them weights. By default α is set to 0.125 and β to 1.25. The predefined target delay q_{ref} is 20 ms:

$$\bar{p}(t) = p(t-1) + \alpha \cdot (\bar{q} - q_{ref}) + \beta \cdot (\bar{q} - q_{t-1}) \tag{2.25}$$

To ensure that PIE is work conserving, we may bypass the random drop if the delay sample $\bar{q}$ is smaller than $\frac{q_{ref}}{2}$, the probability $p(t)$ is not bigger than 0.2 or the queue has less than a couple of packets.

This probability is recalculated every 30 ms, so if a burst is short enough to be transmitted on this interval, PIE will not be affected for it and thus, it wont

penalize finite queue increments. There is also another burst tolerance function that allows bigger bursts if they're under a defined threshold of 150 ms by default.

2.3. Related work

This research have been conducted by using two different engineering solutions designed for different (but tightly related) problems that ends converging, so the very nature of this work had made us work from both sides of the problem. For this reason we would like to remark two important publications, each of them covering one side of the equation.

The first publication we would like to highlight is the excellent survey [1] made by R. Addams about AQM mechanisms which, besides including analysis and definitions for such that algorithms, explains the basic concepts behind them and adds a bunch of useful classification modes by type of congestion indicator, control function, etc. However, this survey does not include some of the last developed algorithms like CoDel or RED. Anyway this document is a very good starting point to gea an overview about the reasons behind AQM dynamics, the reasons why it was firstly created and the diverse approaches that have been used to try to make the queues more advanced.

The second most remarkable publication is the one written by Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock [3] which give us an overall vision about TCP Congestion Control, its characteristics and problems and some of its most important variations. It is without any doubt, a good paper to start understanding such that mechanisms.

There is another important article [76] which puts together AQM and TCP Congestion Control providing a new mathematical mode for time-driven AQM/TCP schemes, that could be useful when evaluating the interaction between these two mechanisms.

There are many interesting articles which study the interaction between TCP and AQM. We have summarized them under these lines and on the table 2.1.

The paper written by Quin Xu et al. [76] offers a new perspective onto the theoretical TCP/AQM analysis field. They claim to outperform the Misra, Gong and Towsley (MGT) fluid model, proposed on [51] and used extensively on analytic AQM analysis, by fixing some inaccuracies provoked by such model like the assumption of independence between congestion window and packet losses or the non distinction between the different TCP congestion control phases. They also reverse the feedback approach using on the MGT model by considering TCP as the feedback provider of AQM. They make use of two extreme scenarios, with the congestion window under the threshold and with it above the congestion window. The results shown, seems to prove that such model fits better to the simulations made on ns-2 [55]. So the analysis made on this paper is just theoretical and its unique purpose is to verify the correct behaviour of the model and neither to test

nor to evaluate different TCP/AQM combinations.

On [60], some TCP versions like Reno, New Reno, Vegas and Tahoe were mixtured with RED, REM, ARED and AutoRED, with the purpose of evaluating the latter. Such simulations were performed on ns-2 [55], using a similar Dumbbell scenario like we did. The results showed a bigger delay on New Reno over the other TCP congestion control protocols on all the different AQM simulations.

The DOCSIS document [74] is basically oriented to a practical implementation of both CoDel and PIE. For this reason, the simulation scenarios are VoIP/Gaming traffic, Web traffic and raw TCP transmissions. They conclude its evaluation stating that, although Fair Queuing approaches like fq-CoDel explained on section 2.2.5 seem to have better performance than single-queue versions, the performance gap is not big enough to justify the added complexity that represents Fair Queuing AQM algorithms. Therefore, they support PIE cause it shows a decent performance and it is simple enough to minimize implementation costs.

The team from the Institute of Communication Networks and Computer Engineering (IKR) of University of Stuttgart, Germany [67] gave also some recommendations about the use of ARED, CoDel and PIE. They simulated a single feedback system created by IKR Simlib [37] and tested the aforementioned AQM algorithms together with TCP Cubic, obtaining a best performance with Proportional Integral controller Enhanced than with the other AQM algorithms.

On the paper [41], a simulation of CoDel, fq-CoDel, PIE and ARED was performed using a real dumbbell topology and the Linux implementation of TCP SACK. Instead of trying to simulate a real TCP traffic, they sent raw TCP data between the endpoints, driving the bottleneck into light, moderate and heavy congestion scenarios. The results shown better performance on PIE over CoDel under heavy congestion, which derived on several recommendations about CoDel's default values. They also concluded than, though ARED seems to be outdated, its metrics shows not so difference with the other new AQM algorithms, so it should be still taken into account.

The following study, [45], is an extensive evaluation of TCP/AQM systems, with special remark on neuron-based AQM schemes, as it own title states. Together with four more conventional AQM algorithms like ARED, PI, IAPI or REM, four neuron-base active queue management mechanisms like Neuron PID, AN-AQM, FAPIDNN and NRL are evaluated. For that purpose, they simulated on ns-2 with TCP Reno both a simple Dumbbell scenario with a single bottleneck link and a complex topology with several groups of computers, routers and bottlenecks. They mostly took into account queue length and the time it takes to converge to a steady state as the main parameters to evaluate the different AQM mechanisms. As conclusions they declare the superiority of the neuron-based schemes, exhibiting smaller queue length, less jitter, faster queue convergence and better adaptability than the more traditional Active Queue Management logarithms.

Kathleem Nichols and Van Jacobson also signed a very interesting article [53] which offers a very good and accesible explanation about router queues and bufferbloat. They tested CoDel and RED using both Cubic and New Reno over a simulation created in ns-2 [55]. They also simulated a Wireless scenario. On all of the experiments, CoDel outperformed RED in both RTT and throughput. This paper is highly recommended, not only for the experimental results but for the excellent explanation of the bufferbloat problem.

On the last paper [11] we included on table 2.1, the authors offer a description about the TCP-friendliness concept as well as a brief survey about router-based schemes to address that problem. Although this work does not include any AQM algorithm related to our work, the analysis of TCP friendliness and how it is affected by AQM is very interesting. The experimental setup utilized was a ns-2 scenario following a Dumbbell topology with two routers connected between them with a link that acts as the bottleneck of the simulation. Attached to each one of the routers are one UDP source and four TCP sources, transmitting 1 kilobyte packet size data. They took special interest to the number of flows and the relations between them. The conclusions mainly addressed the problem of how to detect unresponsive flows and how to protect the other flows against them.

Another useful resource that offers an in-depth overview of the mathematics needed to theorize about Congestion Control algorithms is the fantastic book *Analytical Methods for Network Congestion Control* [46] which shed light on Congestion Control modelization, Equilibrium point characterization and stability analysis.

As you can extract from all the documents reviewed, there has not been a wide study examining the behaviour of the most important AQM schemes under the most commonly used TCP algorithms. Most of the articles refers to two or three AQM algorithms and test it with one TCP algorithm, and there is not a common set of basic characteristics among these different papers so it is very hard, or even impossible, to compare them. This is something that this thesis tries to fix, examining the relationship between Active Queue Management and TCP by using a suite defined by the Internet Congestion Control Research Group with well defined scenarios in order to facilitate the replication of our results.

Table 2.1: Related work

Title	Year	AQM	TCP	Analisis
[76]	2015	PI RaQ REM	Reno	ns-2 and math. model
[60]	2015	RED ARED REM	Reno New Reno Vegas Tahoe	ns-2
[74]	2014	CoDel CoDel-DT SFQ-CoDel PIE SFQ-PIE Tail-Drop	New Reno	ns-2
[67]	2014	ARED CoDel PIE	CUBIC	IKR Symlib
[41]	2014	CoDel PIE ARED SFQ-CoDel	TCP SACK	Laboratory Setup
[45]	2014	ARED PI IAPI REM Neuron PID AN-AQM FAPIDNN NRL	Reno	ns-2
[53]	2012	CoDel RED	CUBIC Reno	ns-2
[11]	2004	CHOKe RED SFB BLACK CARE	Reno TRFC	ns-2

Chapter 3

Experimental Setup

In this chapter we are going to explain the experimental setup we used in order to test TCP congestion control algorithms we already explained together with AQM we talked about on previous sections. For that purpose, we modified the Internet Congestion Control Research Group (ICCRG) TCP evaluation suite [36] to address our needs, by adding such that AQM algorithms.

This suite was created in order to offer the researchers a ns-2 simulation environment ready to quickly evaluate and compare TCP modifications by using a group of common well defined scenarios with specific characteristics, such as data centers, transoceanic links, wireless networks, satellite communications or dial-up links. It is designed to quickly collect metrics like end-to-end delay measurements, goodput, average packet loss, etc.

We decided to use the basic Dumbbell topology provided by that suite. This topology consist of 6 nodes, 3 of them sharing a router that is connected with another router that provides connectivity to the remaining three nodes as it is illustrated on figure 3.1. We changed the RTT and link capacity of each of the links composing this topology in order to simulate the distinctive properties of each of the scenarios we tested, that is, Satellite links, Transoceanic links and Data Center networks.

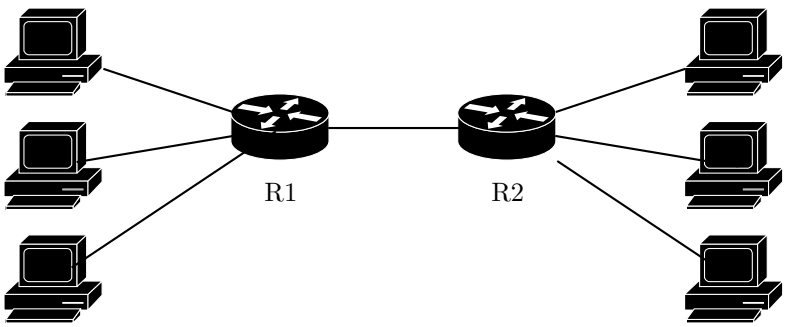


Figure 3.1: Topology of the experiment

The main routers are the ones that we are going to be monitoring, and thus, the link between them would be the one in which we will induce congestion. Each

one of these nodes gathers information about the arrived, departed and dropped packets as well as the average queue size and the traffic load which is the ratio between sent bytes and link bandwidth. Besides these metrics we also observed the time spent by the packets to go thought the queue.

The traffic traces we use are the ones already implemented on the ICCRG suite. Such that traces are based on traffic captured at the University of North Carolina and have a pretty low reverse path traffic, approximately a 10% of the total load, so we will analyze specially the results seen on the direction that supports the biggest load. Approximately 10% of our traffic use 536 byte packets and 90% consist of 1500 byte packets.

We have used the TCP variations included in such that suite because we think they represent a major part of the TCP schemes that are now in use in the world. These variations are New Reno, Cubic, CTCP and LedBat. We made additional tests with New Reno, Cubic and CTCP with an initial window of 10 MSS instead of 3 MSS. Approximately, an 86% of our connections are under 4380 bytes, which is the current standard initial window size (iw3). For that reason we think that the impact of changing the initial window to a bigger one will be limited on this specific simulation setup.

We made simulations with these TCP algorithms together with the Active Queue Managers under various loads and configurations. In the case of Tail Drop and REM we changed the size of their buffers to 25%, 50%, 100% and 200% of its default value of 840Kb. With ARED, CoDel, fq-CoDel and Codel-Dt we modified their intervals from 25 ms to 200 ms going through 50 and 100 ms. We executed these simulations at 85% and 110% of the bottleneck load. The parameters we used for each of the AQM algorithms are depicted on table 3.1.

Buffer manager	Static parameters	Variable
Tail-Drop		$q_{size} = 85, 170,$ $425, 850, 1700Kb$
REM	$\gamma = 0.001$ $\phi = 1.001$ $\alpha = 1$	$pbo = 85, 170,$ $425, 850,$ 1700 Kb
ARED	$\alpha = 0.01$ $\beta = 0.9$ $p_{max} = 0.5$	$t_{interval} = 25, 50,$ $100, 200$ ms
CoDel et al.	$q_{target} = 20$ ms	$t_{interval} = 25, 50,$ $100, 200$ ms
PIE	$q_{ref} = 20$ ms $\alpha = 0.125$ $\beta = 1.25$ $t_{upd} = 30$ ms	$t_{upd} = 7, 15,$ $30, 60, 90,$ 180 ms

Table 3.1: AQM Simulation parameters

The simulation process has several phases. In the first one, the start times of the connection vectors on the original traces are shuffled using a Fisher-Yates shuffle in order to get a stationary load during the whole simulation. When this phase has ended the simulation starts a prefilling phase whose objective is to quickly arrive to a state in which the load on the link is stable. After that phase begins the warm-up phase in which throughput and average queue size are used to confirm that the simulation has reached an "Steady State". When this phase ends, the data collection begins. The data obtained during the simulation are processed later by using Jupyter Notebook [59] which allow us to quickly visualize the gathered traces and dynamically play with it.

The data given by the ICCRG suite consist on two types of files: summaries and traces. Inside summary files we can find the value of a series of parameters per router like the overall throughput, the total number of packets received, the total number of packets dropped, the average queuing delay as well as the average queue size during the simulation and, finally, the average packet loss. The data stored on the trace files are traces of some important values over time, per router. Such values are: arrived and departed megabytes, percentage of packet dropping, link load (the ratio between the data sent to the bottleneck link and its configured capacity), and the average queue size. The granularity of such traces is 0.3 seconds. We also extracted as an extra vector, the queuing delay experimented on each router, over time.

Chapter 4

Access Link

In this scenario, the dumbbell topology we decided to use was configured with the bandwidth and Round Trip Time that appear on figure 4.1. So, according to the findings illustrated by Kleinrock [42], the operating point for this isolated network will be at 100 Mbps of bandwidth and between 4 and 102 milliseconds of RTT.

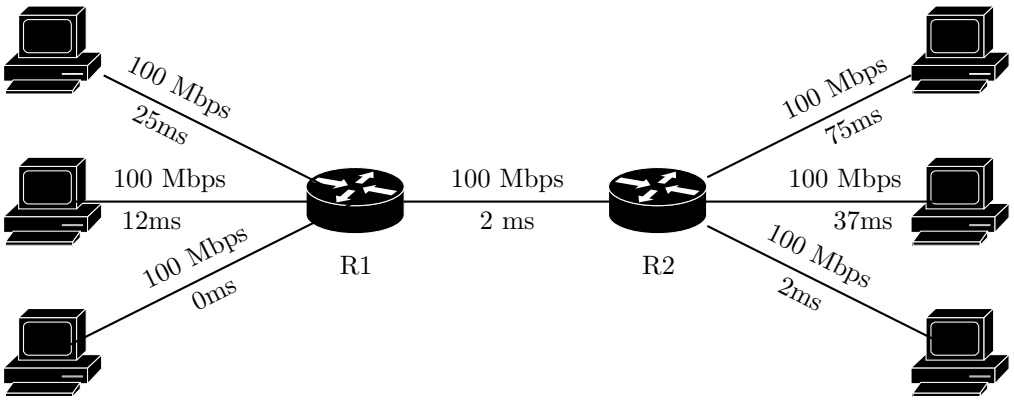


Figure 4.1: Topology of the Dumbbell Scenario

4.1. New Reno

We found that New Reno is an algorithm that works well together with Active Queue Management on this scenario, keeping an average queuing delay that goes from 2.5 to 20 milliseconds as it is shown on figure 4.2 and a packet loss always under the 1% as reflected on figure 4.3. However, link utilization is the worst on our set of experiments, excluding Ledbat.

Both the limited reached load and the low packet loss percentage is due to the low performance of New Reno compared to newer algorithms like CTCP or Cubic more than to an excellent feedback loop created between AQM and TCP New Reno. New Reno performs correctly on networks with low losses and small Bandwidth Delay Product. On this network, BDP is 25000 bytes so according

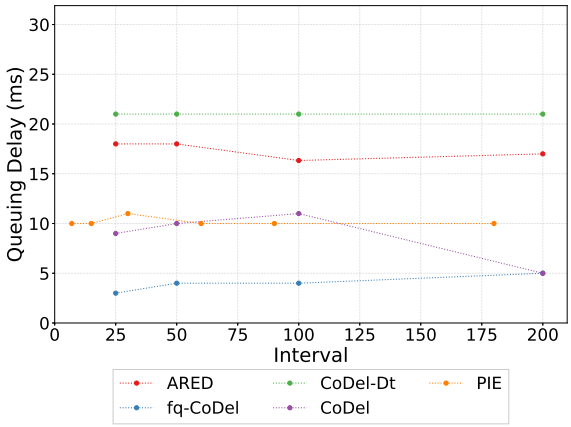


Figure 4.2: NewReno: Delay vs. Interval

to [39] we could consider it a Long Fat Network LFN, that is a network which BDP is bigger than 10^5 bits.

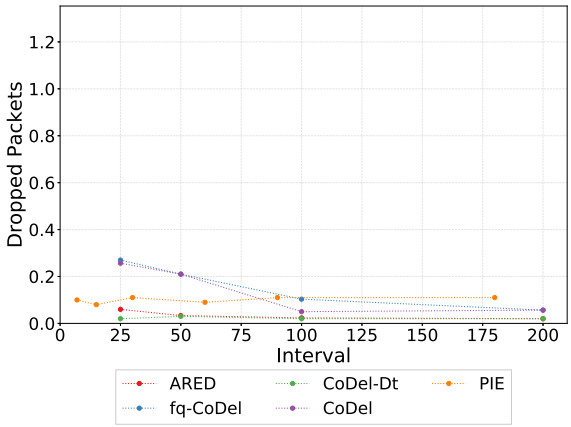


Figure 4.3: NewReno: Loss vs. Interval

If we take a look at the dropped packets on Figure 4.3 and the queuing delay on Figure 4.2 we can also observe how the use of flow separation is helpful in combating bufferbloat: With very similar percentage of packets dropped, Fair Queuing CoDel outperforms generic CoDel by 5 milliseconds in most of the intervals tested. The last interval, in which both algorithms share the same queuing delay and packet drops could be caused by inactivity of both algorithms due to the big update interval, which avoid activating Codel's dropping mode.

4.2. Compound TCP

CTCP presents an increment on the load of a 2% with respect to New Reno (figure 4.7) in exchange for a generalized growth on the delay as it appears on

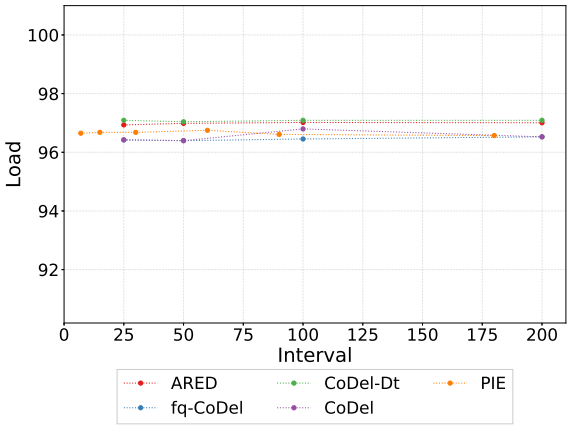


Figure 4.4: NewReno: Load vs. Interval

figure 4.5. Nevertheless, AQM methods like fq-CoDel, CoDel or PIE manage to keep the queuing delay over their 20 milliseconds target.

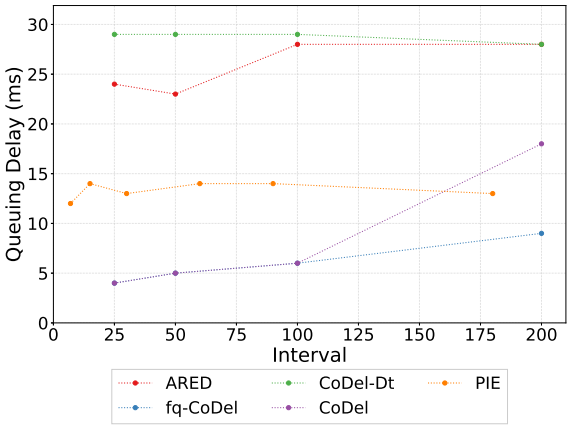


Figure 4.5: CTCP: Delay vs. Interval

As it was explained on Compound TCP draft [71], delay-based component added to this algorithm makes it more capable to deal with packet losses when link delay is low, which enables it to reduce the time required to reach full link utilization after a loss event and to improve the performance over New Reno specially when delay on the link is lower enough to let the delay-based congestion window grow. As this additional window is configured to not to be negative, CTCP is lower bounded by its loss-based congestion window and it wouldn't show an outcome lower than New Reno.

The control function that some of the simulated queue managers use, which is to periodically dropping packets grant to the special feature of CTCP an important relevance. In consequence, queue managers like fq-CoDel or CoDel should demonstrate a fair behaviour.

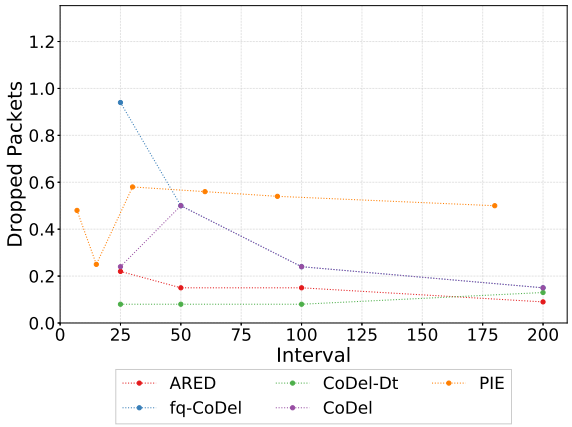


Figure 4.6: CTCP: Loss vs. Interval

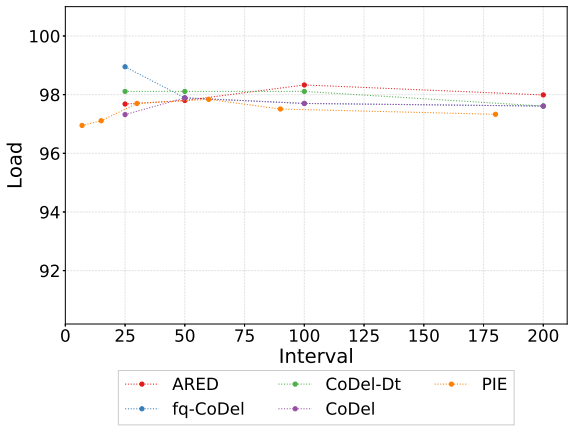


Figure 4.7: CTCP: Load vs. Interval

We observe a reduction on the number of packets dropped under CoDel and fq-CoDel algorithms on Figure 4.6 as a result of the increment on the update interval value. Although at first sight this could be interpreted as a good sign, this situation cause an increment on the queuing delay experimented on the link. This means that bigger intervals between packet drops makes CoDel unable to manage spikes shorter than such interval, thus leading to the creation of a standing queue, falling into the so called bufferbloat problem that has as a result the delay rise we observe on Figure 4.5. That situation supports the adoption of the 100 milliseconds interval as the standard value for CoDel. This is logical cause the RTT on this scenario falls into the range in which CoDel has better performance when it uses 100ms as interval, according to [54].

It is interesting also to remark the steady behaviour of PIE in terms of delay across all its tested interval values. It is also remarkable, the decrease on the percentage of dropped packets when its update interval falls under the 30ms default value. It seems to suggest that a value closer to 15 milliseconds should reduce packet loss while keeping a short enough buffer size.

4.3. Cubic

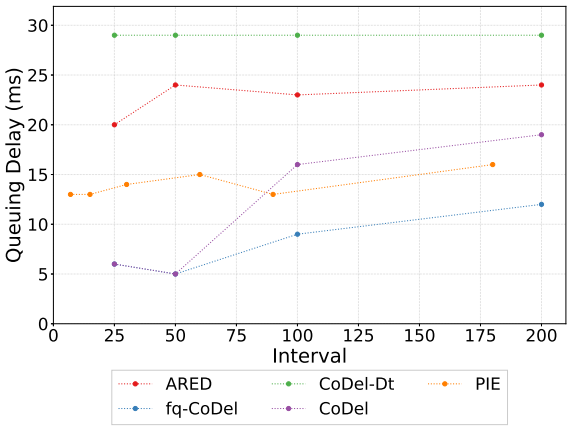


Figure 4.8: CUBIC: Delay vs. Interval

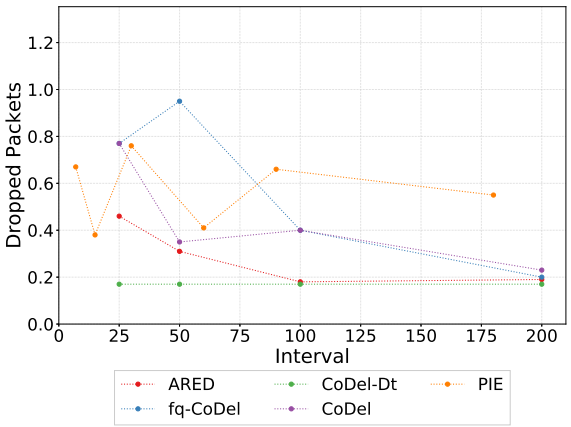


Figure 4.9: CUBIC: Loss vs. Interval

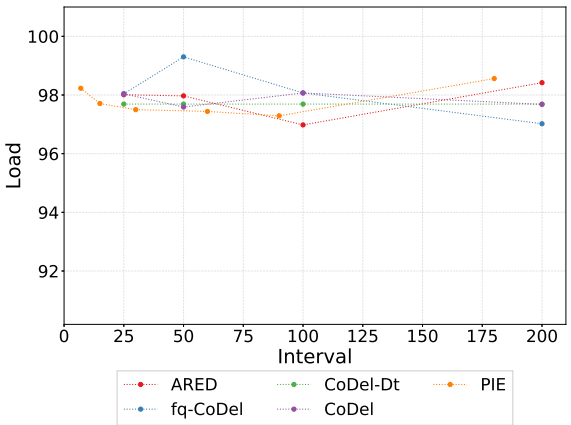


Figure 4.10: CUBIC: Load vs. Interval

Cubic is an algorithm designed for Long Fat Networks too, but unlike CTCP, it just relies on packet loss for detecting congestion so it is supposed to have a different interaction with AQM. Nonetheless, it is designed to provide fast recovery of the congestion window after a loss event so it should reflect a performance similar to CTCP.

Under this TCP algorithm we notice a very light load rise with respect to both CTCP and New Reno (Figure 4.10). This phenomena also produces an increment on the percentage of dropped packets and on the queuing delay. The relation between update intervals, percentage of dropped packets and queuing delay we talked about on the previous section can also be observed on this simulation in which some of the algorithms like PIE, CoDel, CoDel-DT or even ARED follow such that trend on 4.9

We must mention the bad behaviour of CoDel-DT, here and also under the previous TCP algorithms. Few literature about this modified version of CoDel was found, but it seems clear that the congestion indicator used it is not correctly estimating the amount of delay that is present on the link, considering that the congestion indicator is the only thing that changes between this algorithm and the original CoDel. Adaptive Random Early Detection, although it is one of the simulated algorithms which collects more queuing delay (Figure 4.8), it also presents a steady conduct over all the intervals tested. This lead us to think than, as this algorithm uses as congestion indicator the queue size, such value is not low enough to allow fair comparison with the other algorithms which use target delay.

The oscillating behaviour across intervals that PIE shows may be due to a counterproductive synchronization between Cubic and PIE. It is possible that the AQM algorithm would be acting against the fast cubic window growth function implemented on this TCP flavour, although this should only occur when buffers on the bottleneck are full and Cubic produces a sudden reduction on the departure rate.

The way most of the algorithm converges on the 200 milliseconds interval lead us to think that under that configuration none of the Active Queue Management algorithm is really acting against congestion or they are just weakly acting against it.

4.4. LedBat

Ledbat obtains nearly 0% packet drops as it is shown on figure 4.12 and a queuing delay lower than 5 milliseconds on every AQM scheme we have tested (Figure 4.11). Notice that 5 milliseconds is the self-imposed delay threshold that LEDBAT uses by default. The inner design of this particular protocol also has a negative impact on bandwidth utilization, being the lowest of all the simulations we have performed. Let's do not forget that we are simulating just one TCP algorithm at a time. On a simulation with multiple TCP algorithms running several parallel connections LEDBAT could suffer starvation produced by other competing TCP

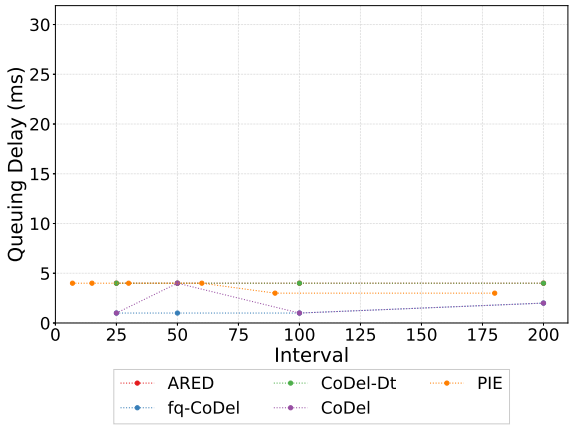


Figure 4.11: LEDBAT: Delay vs. Interval

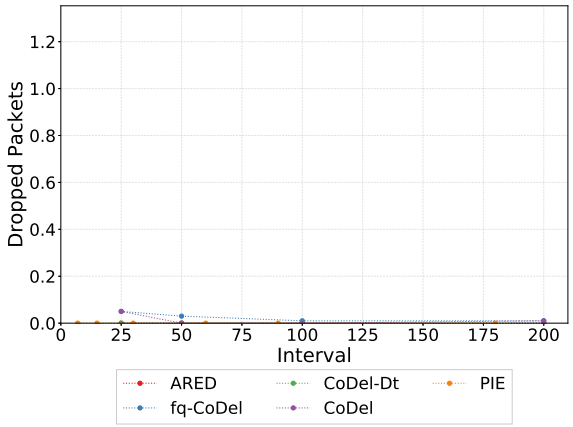


Figure 4.12: LEDBAT: Delay vs. Interval

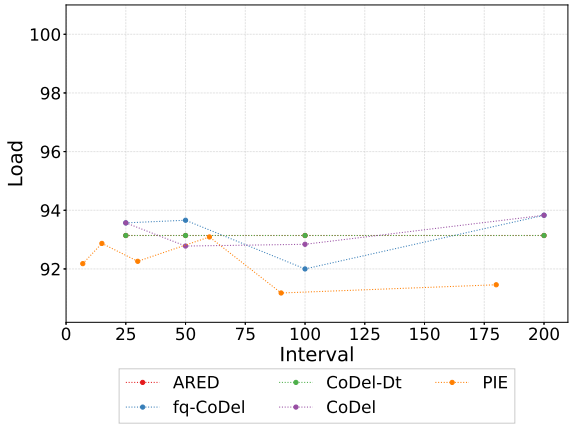


Figure 4.13: LEDBAT: Load vs. Interval

flows, and act much worse in terms of load.

With such level of packet dropping, we think this experiment it is not useful in terms of AQM analysis. Ledbat should take all the credit here for the eradication

of the bufferbloat.

Chapter 5

Datacenter Scenario

In this scenario we tried to simulate a characteristic links inside a datacenter. These networks count with high capacity links from 1 Gbps onwards and very small delays of the order of microseconds. Although we have utilized the distinctive RTT and bandwidth parameters of a datacenter, we keep the topology untouched so to get focused on the analysis of the bottleneck link and to keep coherency over our whole work. Some studies like [5] and [29] suggest the use of more complex scenarios that could offer a more complete and comprehensive view of this particular network configuration.

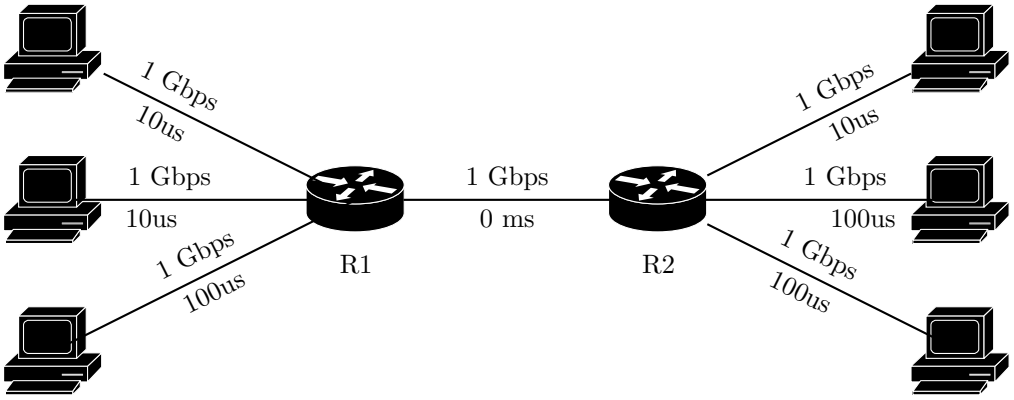


Figure 5.1: Topology of the Datacenter Scenario Scenario

There has been some discussion the last years regarding datacenters, almost focused on how to address some specific problems these scenarios used to have like TCP Incast, which is the throughput collapse that occurs when multiple sender tries to transmit data simultaneously to a single receiver. Although it seems that TCP Incast is a transport layer problem, we think it is truly originated on upper layers of the TCP/IP stack due to the specific traffic generated on such those scenarios. This simulation uses the same traffic traces like the other ones, where the traffic is unsynchronized and does not correspond to the traffic generated by distributed nodes. That is the reason why our datacenter scenario shouldn't suffer this specific type of problem and shows very acceptable delays.

It is interesting to remark, though, that the interactions generated between TCP and AQM are still significant, as we'll see now, and interesting from our point

of view. One some other scenarios, the AQM and TCP algorithms are tested on moderate to heavy latency conditions. We think it is appealing to observe what happens when conditions are just the opposite.

5.1. NewReno

As we can see on Figure 5.2, delay is way under its target of 20ms for PIE, CoDel et al. and the percentage of packet drops is closer to 0% for almost every queue manager tested. The extremely low latency on this scenario makes the control servo operating on New Reno faster than any of the AQM managers tested, which have no time to act against any queue, thus having minimal impact under these circumstances.

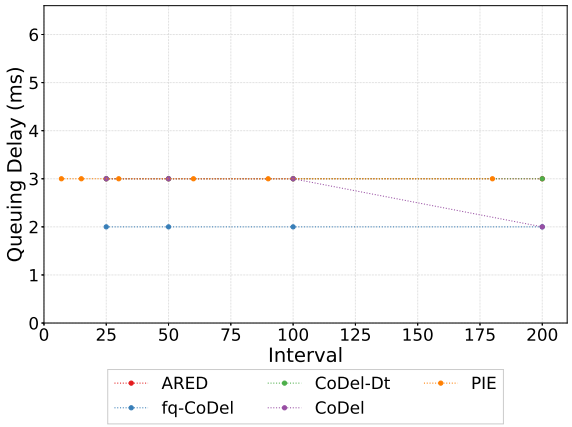


Figure 5.2: NewReno: Delay vs. Interval

A special mention to fq-CoDel has to be made since it is the only algorithm that is able to consistently reduce its queuing delay. We think than the queue management, specially at the enqeue time [34] makes this algorithm act better than any other on this scenario, optimally choosing the dropping interval for each flow and adjusting New Reno tightly, without generating any noticeable rise on the packet loss rate.

5.2. Compound TCP

As it is shown on Figure 5.5, CTCP, specially designed for Long Fat Networks, generate more delay on the studied link, offering an improved load on the bottleneck link in return.

Although the load on Figure 5.7 remains steady across ever tested intervals and algorithm, the same cannot be said for the delay, which increases linearly with the intervals tested on CoDel so as the dropped packets on Figure 5.6

Far from meaning a bad operation of the related AQM mechanisms, this effect

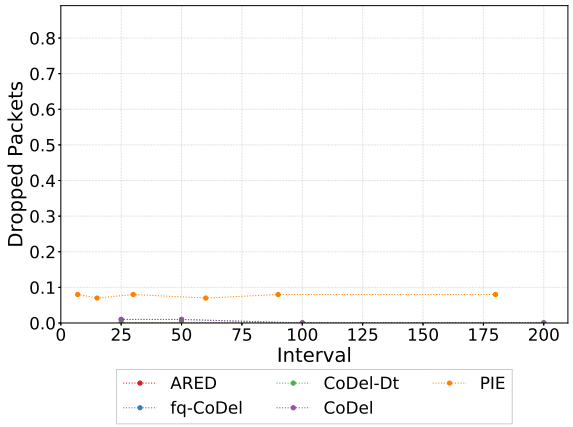


Figure 5.3: New Reno: Queue vs. Interval

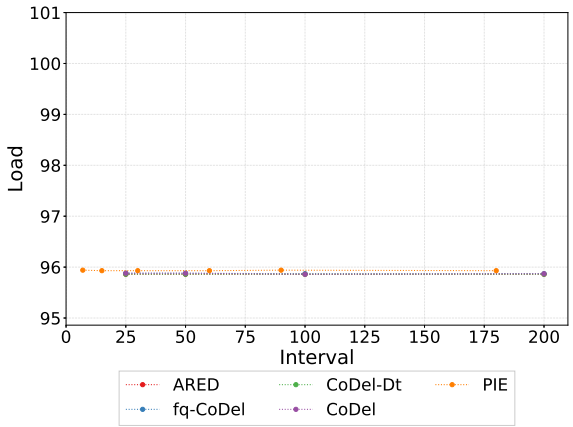


Figure 5.4: New Reno: Delay vs. Interval

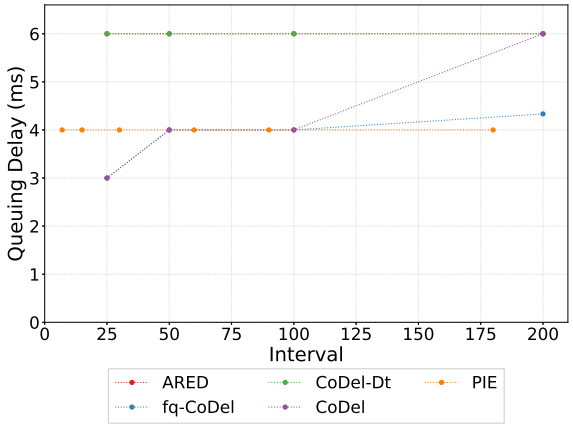


Figure 5.5: CTCP: Delay vs. Interval

is caused by the lost of management created by the rise on the dropping interval. As the aforementioned graphs show, a loosening on the control intervals does not have a positive reflection neither on the load nor on the percentage of packets dropped on the bottleneck link, as we said on previous sections. The small Round

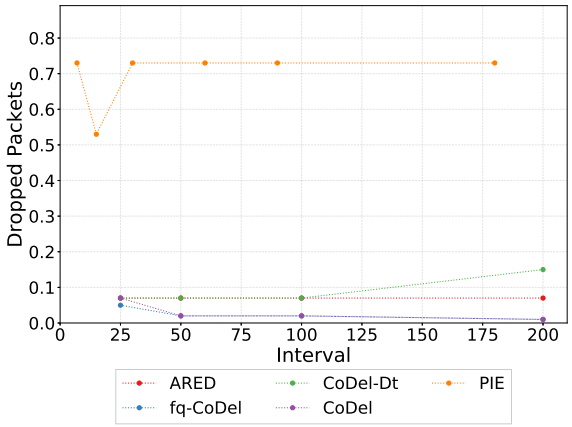


Figure 5.6: CTCP: Loss vs. Interval

Trip Time, though, helps to minimize the impact of such changes on the update interval.

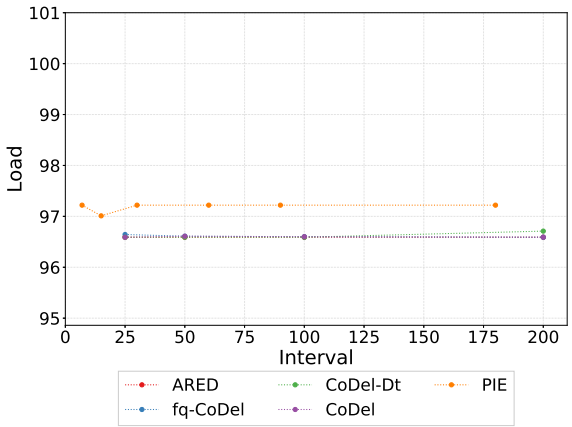


Figure 5.7: CTCP: Load vs. Interval

The characteristics of this network fully enable the delay based component added to CTCP, thus improving the link utilization, although such feature works better on networks with higher RTT, where other algorithms have difficulties growing its congestion window. The large number of packet dropped by PIE, remarkable bigger than any of the other algorithms, get us to think that such algorithm is very sensitive to the nominal latency of a network, regarding its update interval.

5.3. Cubic

Cubic presents very similar results as its counterpart CTCP. One interesting fact that needs to be mentioned again is the unexpected behaviour of PIE, which shows an unusual percentage of packet dropping as it is shown on Figure 5.9, seven times greater that its value under New Reno. We observed a similar conduct on

the preceding Congestion Control algorithm and even on the previous Access Link scenario, but it was considered within the realms of what was reasonable because CoDel showed a similar trend when using interval closer to the 30 milliseconds interval that PIE utilizes to update its congestion indicator and its queuing delay was sticked to its target value of 20 milliseconds.

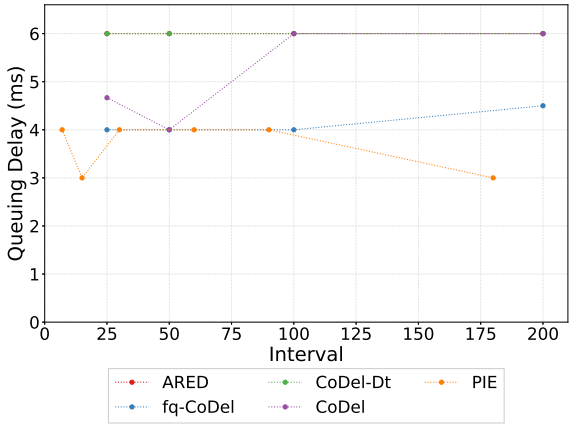


Figure 5.8: Cubic: Delay vs. Interval

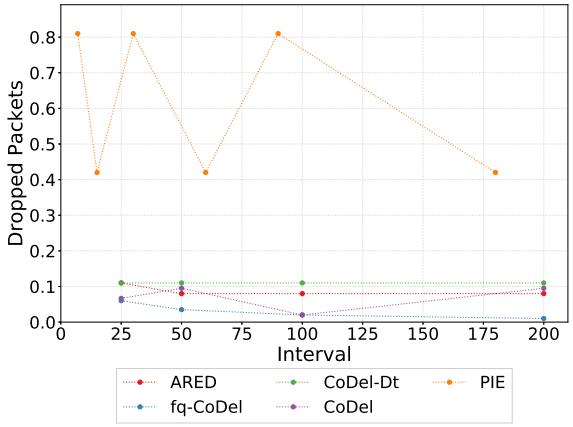


Figure 5.9: Cubic: Loss vs. Interval

This oscillation on the packet dropping percentage has a direct effect on the load, as it is shown on Figure 5.10. As it is recommended on [57], the update interval should be under 15 milliseconds and the α and β parameters should be increased accordingly in order to get a smoothed response, which means that PIE needs special tuning when used on high throughput networks. We should also mention that, besides the high variation on delay, PIE also gets the lowest delay on this scenario as it is displayed on Figure 5.8.

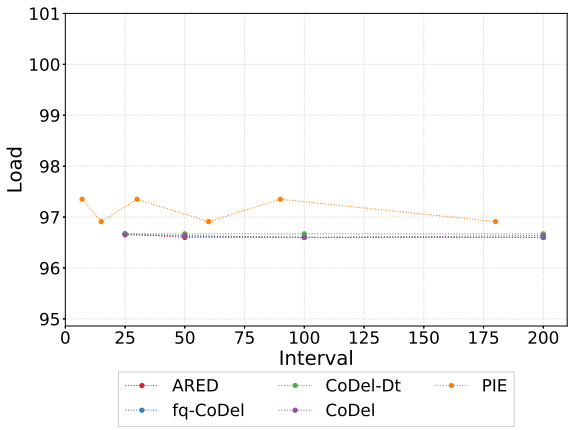


Figure 5.10: Cubic: Load vs. Interval

5.4. LedBat

LedBat is able to maintain queuing delay over its target value of 5ms (Figure 5.13), while reaching load bigger than New Reno and slightly below CTCP or Cubic as it is represented on Figure 5.11. Such improvement over New Reno is due to the low latency experimented on this scenario. Ledbat takes advantage of it, keeping a good queue utilization, without inducing extra buffering. It is milder here, but we can also notice the strange behaviour of PIE also on this scenario, specially on Figure 5.12.

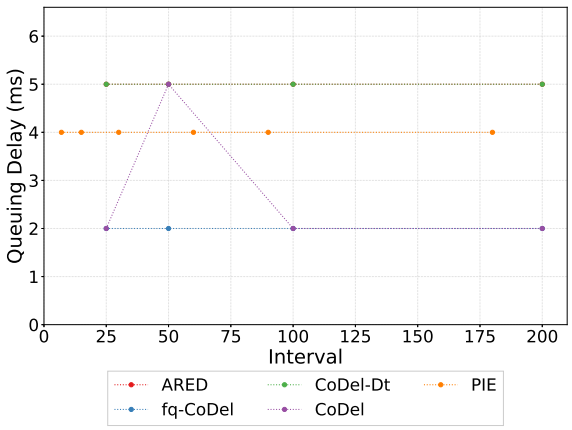


Figure 5.11: Ledbat: Delay vs. Interval

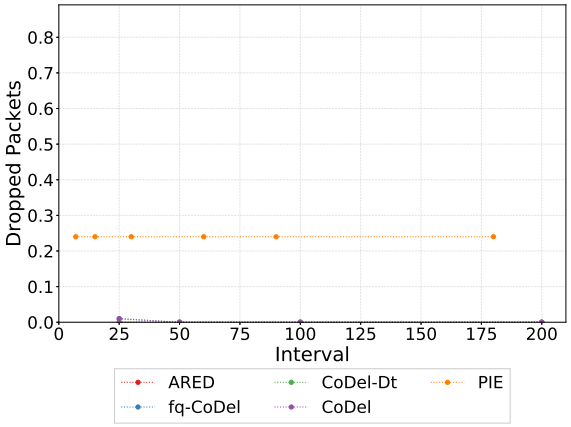


Figure 5.12: Ledbat: Loss vs. Interval

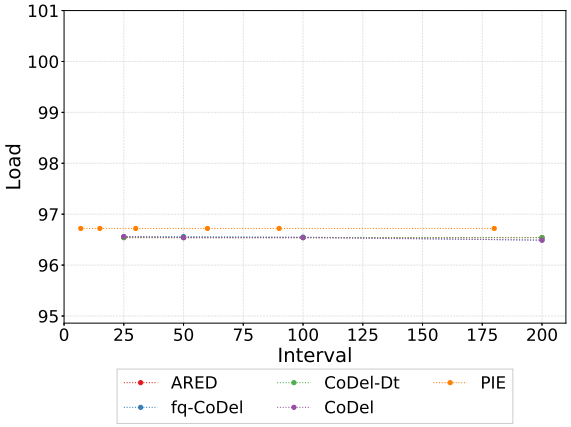


Figure 5.13: Ledbat: Load vs. Interval

Chapter 6

Satellite Scenario

In this section we show the results of a simulation over a different scenario where the main link have the characteristics of a satellite link, with a restricted up-link and a much wider down-link and suffering large RTT's.

In this particular scenario, though most of the traffic flows from left to right as we stated on chapter 3, the main congestion problem is going to take place on the uplink connection, where traffic rates greatly surpasses the capacity of the link. So the congestion is not going to appear on the link which sends the data but on the link which answers with the acknowledgements. This situation could be interesting because it staged a typical problem, not only inherent to Satellite links, but to other types of asymmetrical connections.

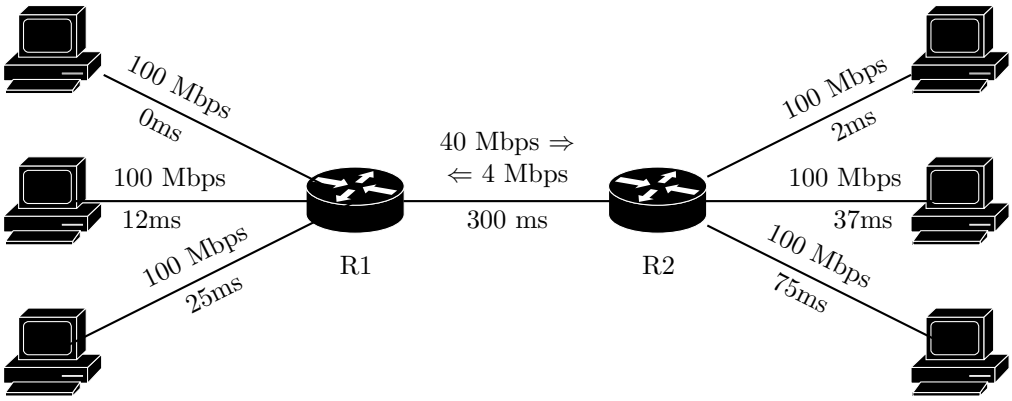


Figure 6.1: Topology of the Satellite Scenario

It is remarkable too the strange behaviour of CoDel-DT. This algorithm not only gets the worst results in terms of delay, it is also the algorithm that drops highest number of packets too. Under our simulations it seems clear to us that CoDel-DT does not reach the objective of maintain the buffers occupancy low enough to reduce queuing delay. What it is interesting in this case, is that this high buffer occupancy it is not caused by a low packet dropping rate.

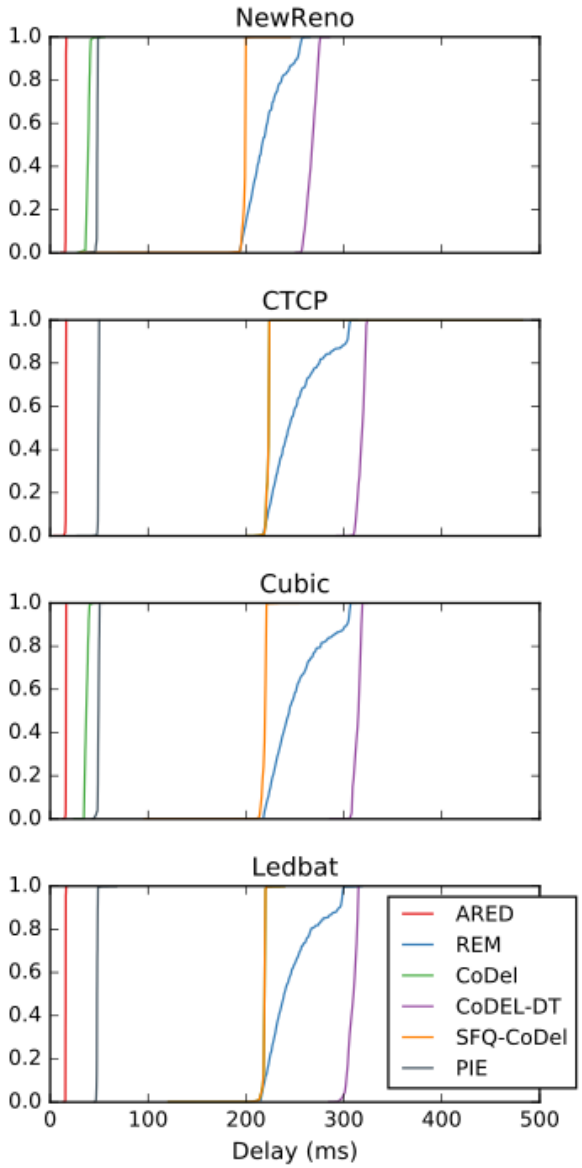


Figure 6.2: Cummulative distribution of delay on Satellite scenario

6.1. NewReno

On our initial explanation of fq-CoDel on section 2.2 we stated that in the worst possible scenario such that algorithm would behave as a single CoDel instance. As we can see on figure 6.3 the delay obtained when using it greatly surpasses the one obtained when we use the original CoDel algorithm.

As we know, if the queuing latency stays over the update interval, CoDel goes into dropping mode. Our theory is that, under this scenario, the fair queuing mechanism utilized on fq-CoDel makes it less sensitive to overall buffer occupancy. The isolation between flows, each one of them ruled by an separated CoDel instance, could make fq-CoDel unaware of the overall buffer occupancy. This could be worse if the traffic going through the bottleneck is composed by short

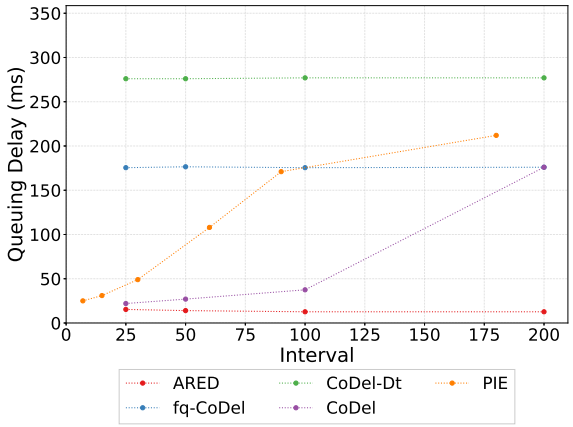


Figure 6.3: NewReno: Delay vs. Interval

connections of just a few RTTs long.

The load values observed on figure 6.5 allow us to get an idea of the congestion on the link, which is receiving from 2 to 5 times its nominal capacity, in other words, it's trying to send from 8Mbps under ARED to 20Mbps with CoDel-DT.

Another surprising outcome is the queue latency and the percentage of packet dropping obtained by ARED that we can see on figures 6.3 and 6.4. Even tough it is the algorithm with smaller percentage of packets dropped, it manages to keep its queuing delay under 25 milliseconds. It seems that the congestion indicator is fully activated on this scenario unlike the previous ones and it manages to keep the queuing delay under 25 milliseconds along all the interval values tested.

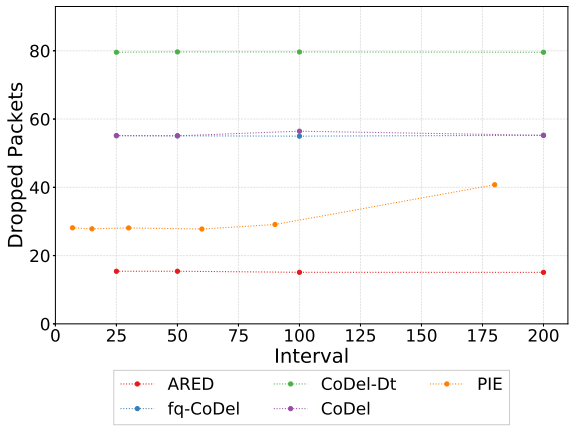


Figure 6.4: NewReno: Loss vs. Interval

It is also remarkable how PIE on its default interval value of 30 milliseconds manages to keep a low delay in comparison with the other algorithms, while its counterparts, CoDel and fq-CoDel are far above such value and inflicting two times more percentage of packet drops on the link, moreover. Regarding fq-CoDel, it

seems to react poorly against heavy congested scenarios, causing a very similar number of packet drops as CoDel, but far away from reaching the same good performance in terms of queuing delay.

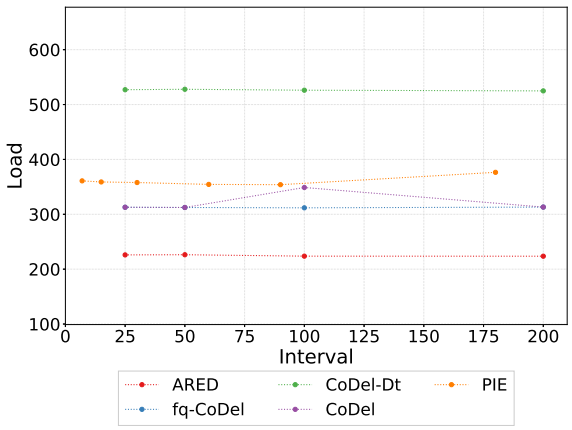


Figure 6.5: NewReno: Load vs. Interval

6.2. Compound TCP

At the delay graph showed here on figure 6.6 we can see that the original CoDel algorithm accuses the change on the interval parameter worse than its counterparts. As the main link on this topology has a RTT of 300 ms, CoDel shows better efficiency with the update interval closer to that value (200 ms).

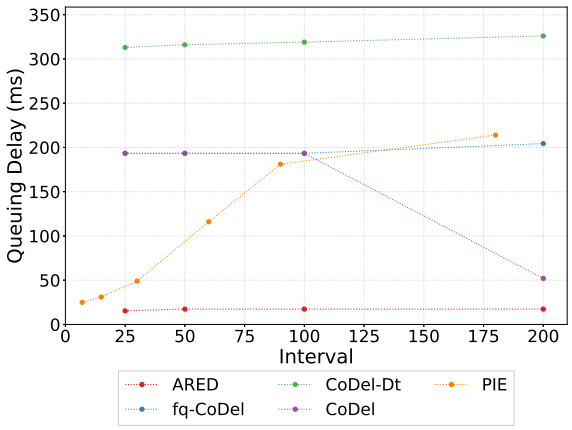


Figure 6.6: CTCP: Delay vs. Interval

It is clear for us that both PIE and ARED respond appropriately to the heavy congestion generated on the satellite uplink, maintaining a low delay (under this circumstances) and an assumable percentage of packet loss as it is shown on figure 6.7. This specific trade-off is not reached by CoDel in any of its modifications under CTCP

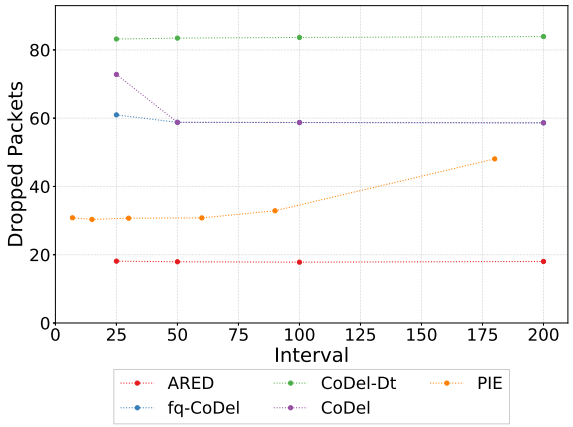


Figure 6.7: CTCP: Loss vs. Interval

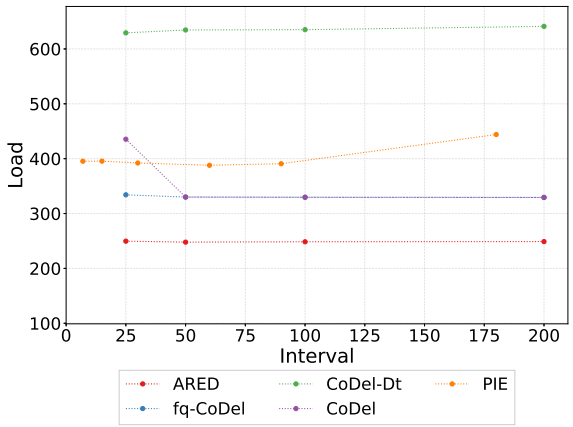


Figure 6.8: CTCP: Load vs. Interval

6.3. Cubic

As we can see on Figure 6.9 and 6.10, Cubic shows metrics very similar to its counterpart, CTCP. Under such overloaded scenario, both TCP mechanisms shares comparable graphs, except for CoDel, which seems to be able to act against bufferbloat at earlier intervals than under Compound TCP, drastically reducing its queuing delay from 200 milliseconds to 40 milliseconds.

6.4. LedBat

The amount of traffic supported by the upload link make us ask ourselves about the suitability of this scenario to test AQM algorithms. While on the download links we have seen loads between 25-60% of the overall capacity of the link, on the uplink connection we have suffered loads up to 10 times superior to the capacity of the link, that is, we have overflowed a link with a capacity of 4 Mbps with 40 Mbps.

As a result of this situation, even LEDBAT, which is a completely different

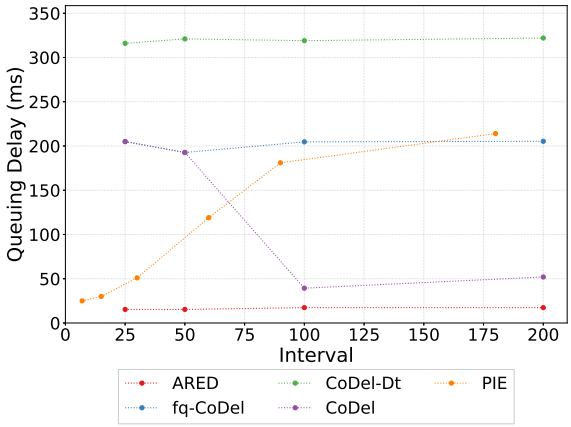


Figure 6.9: Cubic: Delay vs. Interval

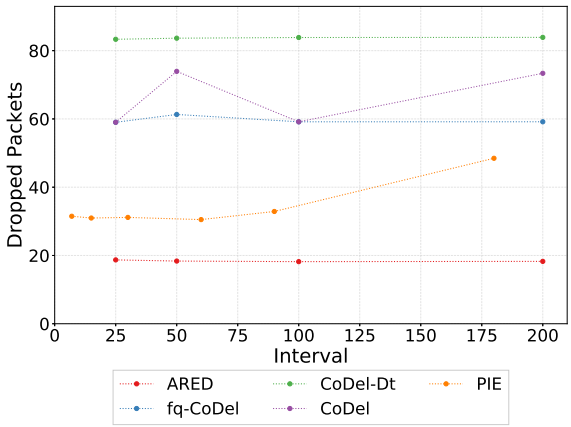


Figure 6.10: Cubic: Loss vs. Interval

algorithm, based on another paradigm to deal with congestion control, displays metrics extremely similar to CTCP and Cubic.

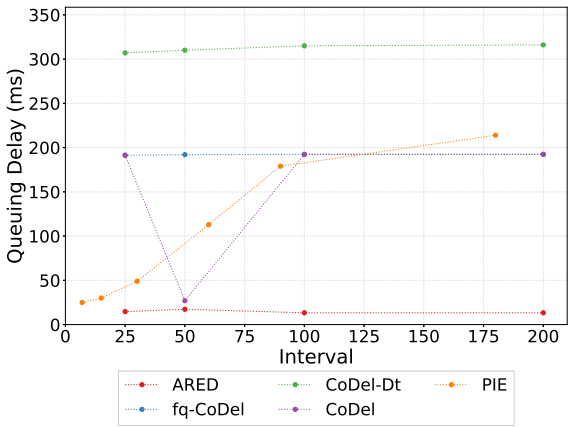


Figure 6.11: Ledbat: Delay vs. Interval

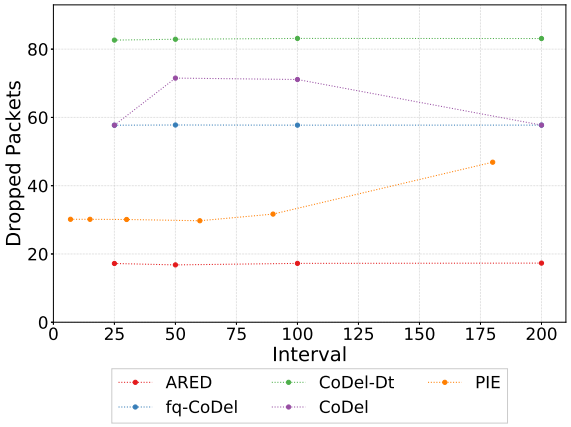


Figure 6.12: Ledbat: Loss vs. Interval

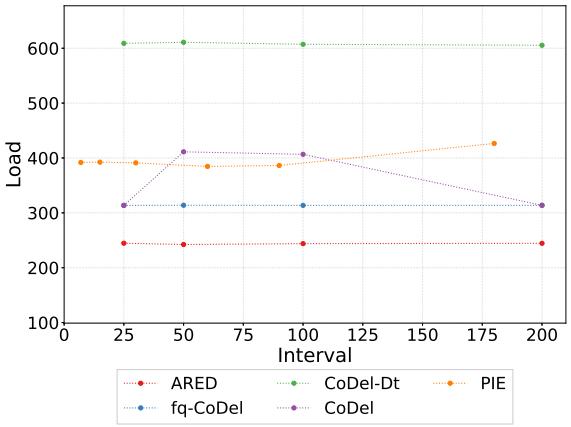


Figure 6.13: Ledbat: Load vs. Interval

Chapter 7

Transoceanic Scenario

This scenario represents a transoceanic link, which give us an excellent opportunity to study the behaviour of algorithms designed to work on high speed and long distance networks, such as CTCP or Cubic, together with AQM mechanisms.

Due to their big propagation delay produced by the long distances this type of links need to cover, they are characterized to offer a poor efficiency in terms of capacity utilization when common TCP congestion avoidance algorithms are in use. Under such those algorithms, in most of the cases the transmission ends even before entering congestion delay phase thus under-utilizing the network capacity. For that reason more aggressive algorithms like Cubic or CTCP were created in order to get a faster convergence towards the link capacity on this specific type of links.

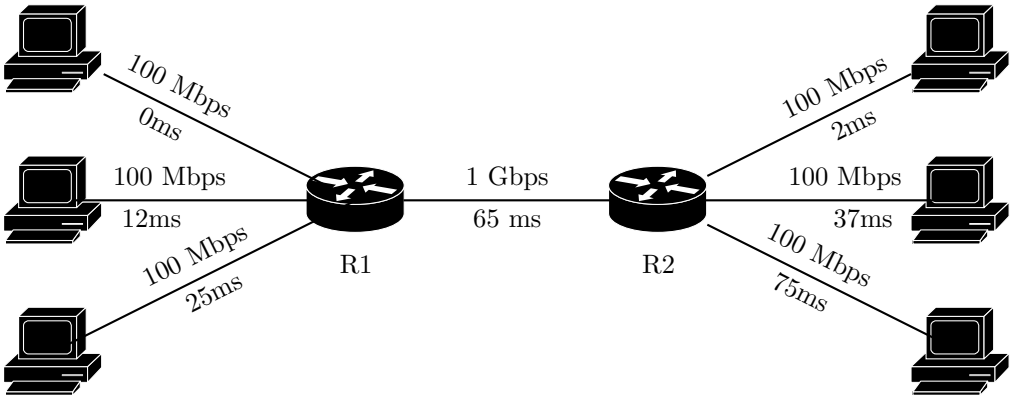


Figure 7.1: Topology of the Transoceanic Scenario

The topology of this experiment reflects the characteristics of such networks by imposing a big RTT on the bottleneck network which is intended to simulate a transatlantic link itself. This values are partially based on the work made by [30] and the latency data gathered by Verizon service provider company [73].

One obvious conclusion drawn thanks to this experiment is that the differences between AQM mechanisms increases with the size and delay of the links under

study. As we can see on figure 7.3 for instance, only PIE is able to maintain delay under the predefined value of 20 milliseconds.

Another interesting thing we can observe on the traces, specially on the fq-CoDel ones, is the clear separation between TCP algorithms. For instance, on figure 7.2 we can clearly notice the separation between the algorithms CTCP and Cubic that include the increased initial window and the ones who have the standard initial window. Just after these two groups, we have New Reno with the increased initial window and the original New Reno algorithm which pays their lower delay with worst throughput than the aforementioned ones. On the bottom we can find LedBat keeping its 5ms targeted delay. There is an increment of more than 10 ms on CTCP and 7 ms on Cubic over the algorithms with the initial window set to 3. The differences on New Reno are less obvious.

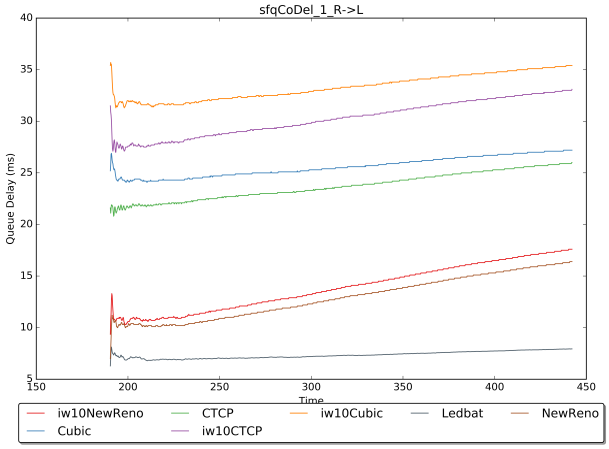


Figure 7.2: fq-Codel: Delay vs. Simulation time

7.1. NewReno

Fair Queuing CoDel is again the best in terms of delay, keeping it under 20 milliseconds with low packet losses as it is displayed on Figure 7.3. It is interesting to observe that, this algorithm is in fact able to obtain such delay metrics with a very similar packet drop percentage (Figure 7.4) than CoDel, which enforces the arguments about the suitability of the fair queuing version over the original algorithm.

On CoDel, as it should be expected, the best results are obtained when the interval is closer to the bottleneck's link RTT. The same does not happen with ARED, which improves its efficacy as interval grow. Lets remember than the update interval of this algorithm is used to refresh the dropping probability, like PIE. On this case, it seems than the time spent between updates positively affects the network load (Figure 7.5) and the percentage of packets dropped (Figure 7.4).

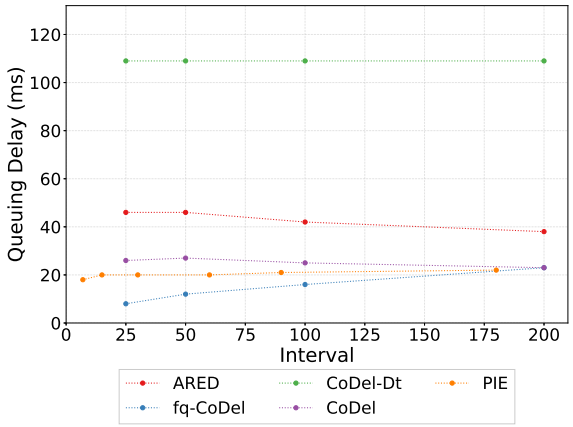


Figure 7.3: NewReno: Delay vs. Interval

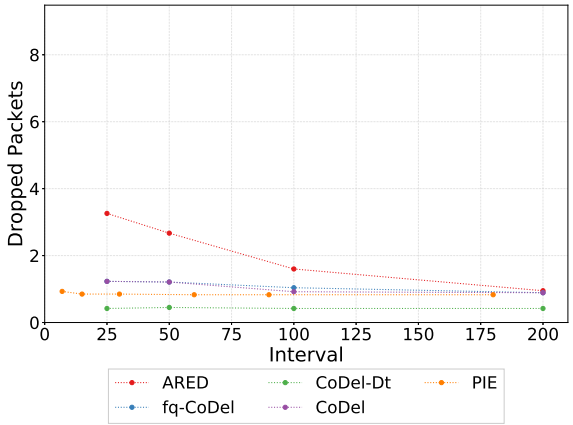


Figure 7.4: NewReno: Loss vs. Interval

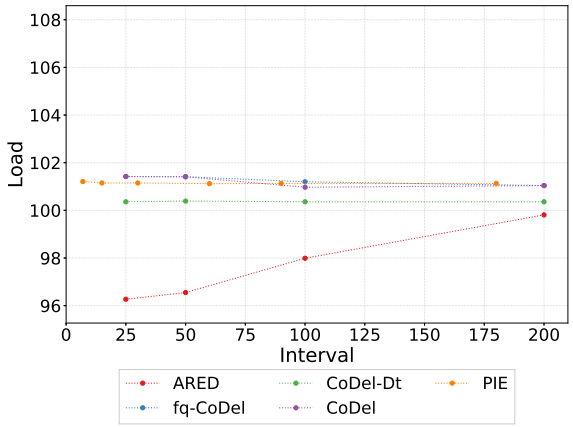


Figure 7.5: NewReno: Load vs. Interval

7.2. Compound TCP

Under this TCP algorithm we can observe a significant rise of packet loss (figure 7.7) and queuing delay (figure 7.6) compared to New Reno. Queuing delay seems

to increase together with bigger update intervals as in the former section and reciprocally proportional to the percentage of dropped packets. However, with this TCP algorithm, such relationships is more obvious. For instance, CoDel goes from 20 milliseconds of delay when using 25 milliseconds as update interval, to 90 milliseconds when utilizing 200 milliseconds as update interval. Bigger intervals reduce the ability to detect bufferbloat and act against it. The same happens with fq-CoDel and PIE although in this last one, the change is almost unnoticeable.

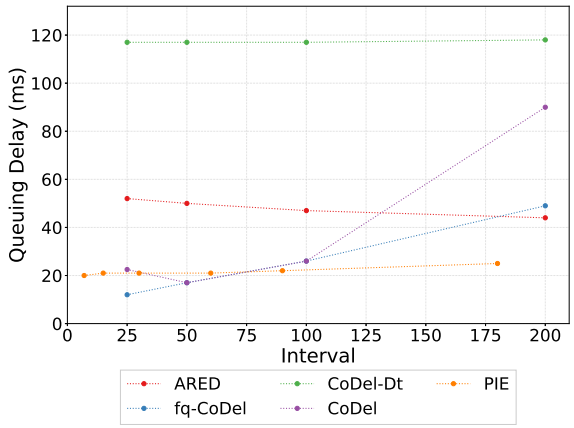


Figure 7.6: CTCP: Delay vs. Interval

We can also see a rise on the number of packets dropped by CoDel-DT, but with no effect on the queuing delay. Notwithstanding being just a minor modification of CoDel, this algorithm have proven itself unable to correctly manage bottleneck queues under any tested scenario.

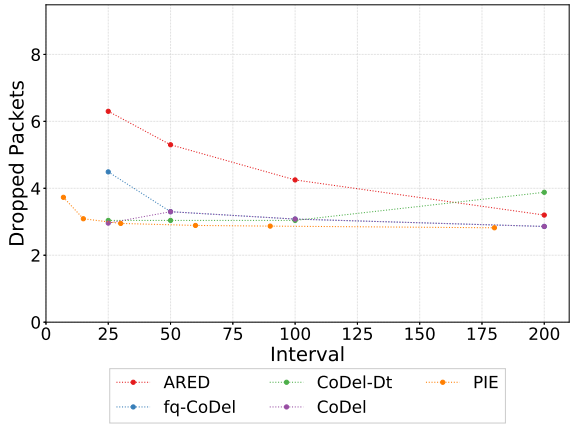


Figure 7.7: CTCP: Loss vs. Interval

7.3. Cubic

The results we've seen previously on CTCP are very similar on Cubic too. In spite of relying just on packet loss as congestion indicator, it displays comparable

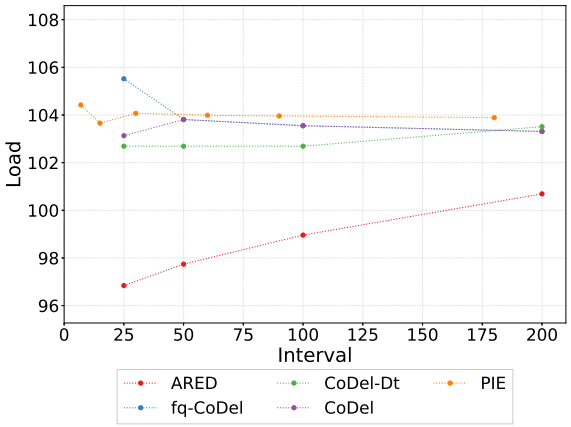


Figure 7.8: CTCP: Load vs. Interval

queuing delay and packet loss rates as it is shown on Figure 7.9. The stability of PIE contrasts with the sensibility of CoDel and fq-CoDel when their interval value is modified. Both algorithms suffer a significant rise on the queuing delay consistent with the rise on the update interval, but the fair queuing variation seems to be more response to these changes and it only increases its delay from 30 to 50 milliseconds, while CoDel experience a delay increment from 30 to 110 milliseconds.

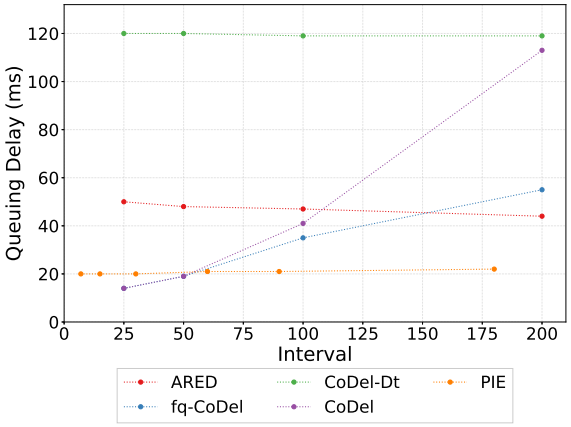


Figure 7.9: Cubic: Delay vs. Interval

As it is shown on Figure 7.11 and unlike on the previous simulation, ARED presents a consistent load across every interval tested.

7.4. LedBat

Ledbat is again a victim on the network characteristics, displaying a queuing delay up to 40 milliseconds even though it was configured to not to surpass 5 milliseconds of queue delay. Regarding AQM, just CoDel-DT surpass the 20 milliseconds target delay imposed. With the exception of this specific algorithm,

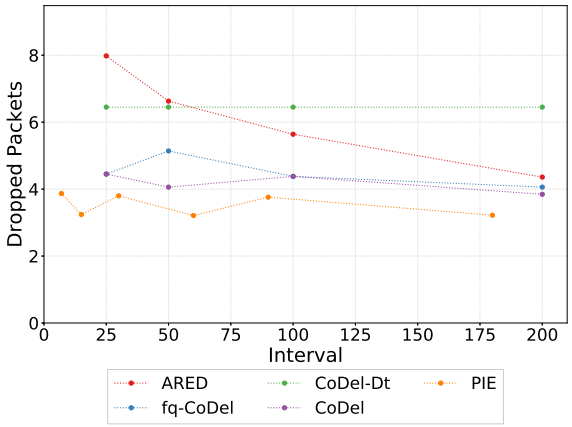


Figure 7.10: Cubic: Loss vs. Interval

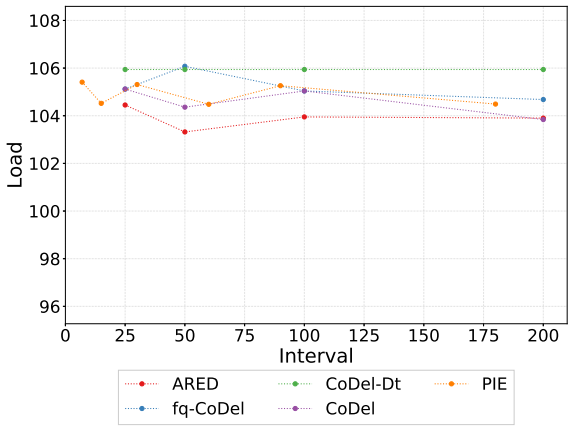


Figure 7.11: Cubic: Load vs. Interval

the rest of them show very reasonable delay, taking into account the scenario, near zero losses and a steady load across all the simulated intervals.

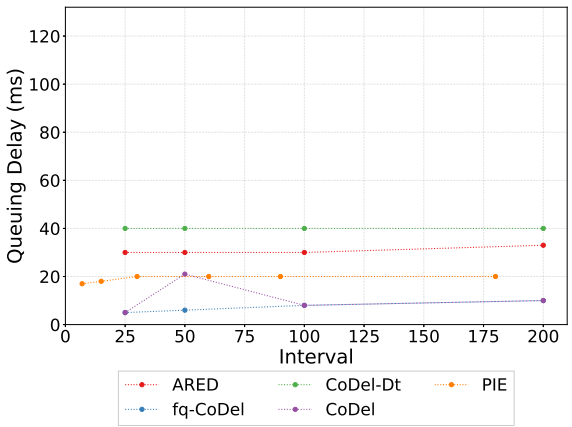


Figure 7.12: Ledbat: Delay vs. Interval

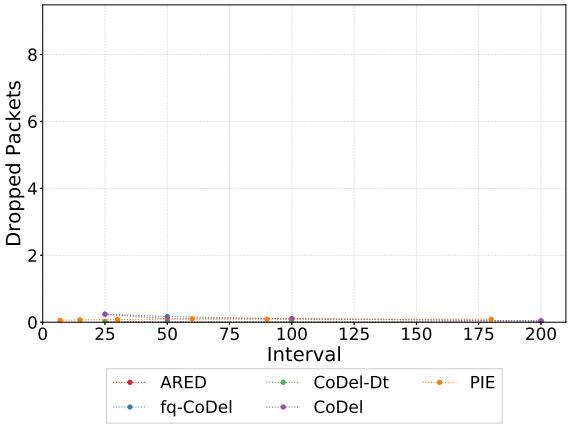


Figure 7.13: Ledbat: Loss vs. Interval

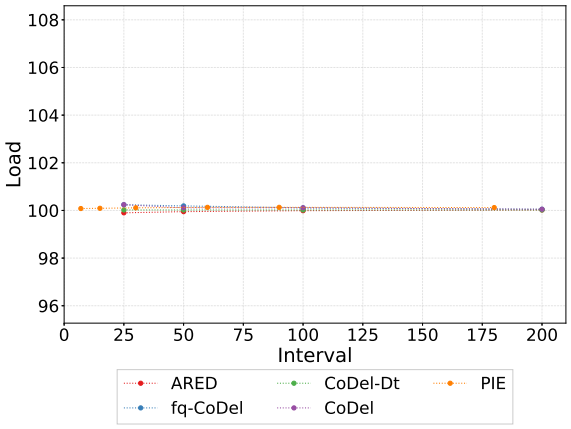


Figure 7.14: Ledbat: Load vs. Interval

Chapter 8

Conclusions

8.1. Conclusions

When Internet was created, the challenge that engineers faced was about how to get maximum utilization on a link, while keeping fairness between different flows. In order to solve that problem, congestion control mechanisms were implemented. Such mechanisms were based on the assumption that when a link has reached its maximum capacity it starts dropping packets. Over time, the rise on the number and size of the buffers across the internet made that assumption a lie. Due to their excessive size and the use of simplistic queue managers, buffers that where there to absorb sudden bursts of traffic were actually persistently filled up with packets. That was called the bufferfloat problem [28].

Bufferbloat delays the detection of congestion, disrupting the proper functioning of the TCP congestion control servo and adding an extra latency due to the standing queue created by the oversized window of the sender. So the logical solution is to properly signal the sender that the limit has been reached and it will create needless buffering if it keeps increasing the window. This is what Active Queue Management does.

We are not just wasting money, having oversized buffers. The delay induced by this problem goes from hundreds of milliseconds to couple of seconds, and such delay can make a difference in a wide variety of applications, from web browsing, which is becoming more and more interactive, to telepresence devices, multimedia streaming applications, multiplayer games and many others.

On this work first we explained the different congestion control approaches that have been used over time, devoting special attention to the algorithms we tested. We also took a look at the AQM schemes used. Then, we simulated several AQM algorithms together with some of the most used congestion control mechanisms implemented on TCP and we analyze them together.

Methods based on queue size are easily predictable: if we increase the size of the buffer or the target size, link utilization increases but queuing delay also does. We can observe this behaviour on REM, for instance if we look at 7.9 and

7.11. But with the new algorithms based on queuing delay, that relationship it is not so clear. Although they reduce the delay experienced on almost any type of networks and under any TCP modification, further study is required to obtain best efficiency in terms of high throughput and low delay. Almost every AQM algorithm is pretty sensible to a large range of variables like configuration parameters, TCP algorithms, network topology and characteristics, traffic details, etc.

An example of this sensitivity could be seen on the behaviour of PIE, an algorithm that shows an steady behaviour across most of the scenarios and congestion control algorithms tested with the exception of the Datacenter scenario, when it shows an unexpected reaction to the low RTT present on this simulation as we can see on figures 5.6 and 5.9.

Except for the aforementioned problems on the Datacenter scenario, PIE offers good delay-loss tradeoff with minimal to none modification in order to work correctly, with the exception of scenarios with high capacity and very low latency, as it was stated on its draft, which recommends to reduce the interval. It has been proven than it works better with update intervals lesser or equal than 15 milliseconds instead of the initially proposed value of 30 milliseconds. Going even further than its own draft, we observe such characteristic on a link with high Round Trip Time and pretty low bandwidth as it is the satellite scenario, in figures 6.3 or 6.6.

CoDel displays good performance in terms on delay and packet dropping, across most of the scenarios and TCP algorithms, although it seems to be outperformed by fq-CoDel as shown in figures 6.6 and 6.9. This modification shows less sensitivity to changes on its update interval as well as the capacity of maintaining queuing delay low, better than most of the algorithms tested, on almost every situation, obtaining more load and less queuing delay without dropping so many packets as others as in figures 6.7 and 6.10. Even changing parameters like its update interval or its target delay does not affect so much to the results as on the single queue version.

In our simulations CoDel-DT offers results consistently worse than any other CoDel variation. This bad performance can be observed since the beginning of the simulations, like 4.2 or 4.5. Its delay prediction could be not good enough as it was thought, at least for conventional networks that are not tied to the requirements of Data over Cable Services (DOCSIS). It is also interesting to observe its relationship with ARED on CTCP and New Reno.

It has been demonstrated that ARED is outperformed by the new algorithms. Besides that, it seems to work better with intervals closer to the bottlenecks link RTT, showing a strong and direct relationship between the interval value and the percentage of dropped packets as it is depicted on figure 7.10. Larger intervals also have an impact on link load as it is shown on figure 7.5. Nevertheless, configuring parameters seems counterintuitive and hard to tune in order to get the best performance. A proof of this theory could be notice on the results given

by such algorithm on the Access Link scenario, Figure 4.2, in which it had a minimal impact maybe because of a loose configuration that did not triggered its congestion indicator.

Almost every queue management approach is able to provide an acceptable capacity in terms of throughput, excluding LEDBAT which shows the worst link load metrics. That is the reason why we think that the presence of AQM schemes could affect significantly to the efficiency of TCP variations that implement delay-based congestion control mechanisms. ECN mechanisms should be adopted in order to assure a correct separation between TCP congestion control and queue management.

8.2. Future work

It could be interesting to extend these simulations to another scenarios like wireless topologies or long fat links, and under different conditions like sudden changes on link capacity or handoff situations. It would be enlightening too the gathering of some other metrics like number of flows, average duration and congestion windows size. To analyze in deep the fairness shown between TCP flows, or the impact on the congestion window growth caused by AQM.

It would be also appealing to simulate the new BBR algorithm designed by Google, as it seems like a promising new way to deal with congestion as well as to maximize link utilization. Some first attempts to analyze this algorithm has been done, some utilizing real hardware [33] or virtualizing it [19]. We also tried to implement BBR on our modified TCP Evaluation Suite [56] but we encounter some problems on the implementation of the TCP Pacing mechanism, essential to the correct simulation of this algorithm.

Out of the simulator, the implementation of the aforementioned scenarios into real hardware could give us a more accurate view of the interactions between this topologies as well as to facilitate the direct experimentation with the existing technologies in use, without having to adapt it to a simulator. A compromise solution could be the use of simulators like Mininet [49]. As it is based on network namespaces, it allows the execution of the algorithm implementation in use on the Linux kernel, giving it great credibility.

Appendix A

Plan and Budget

A.1. Plan

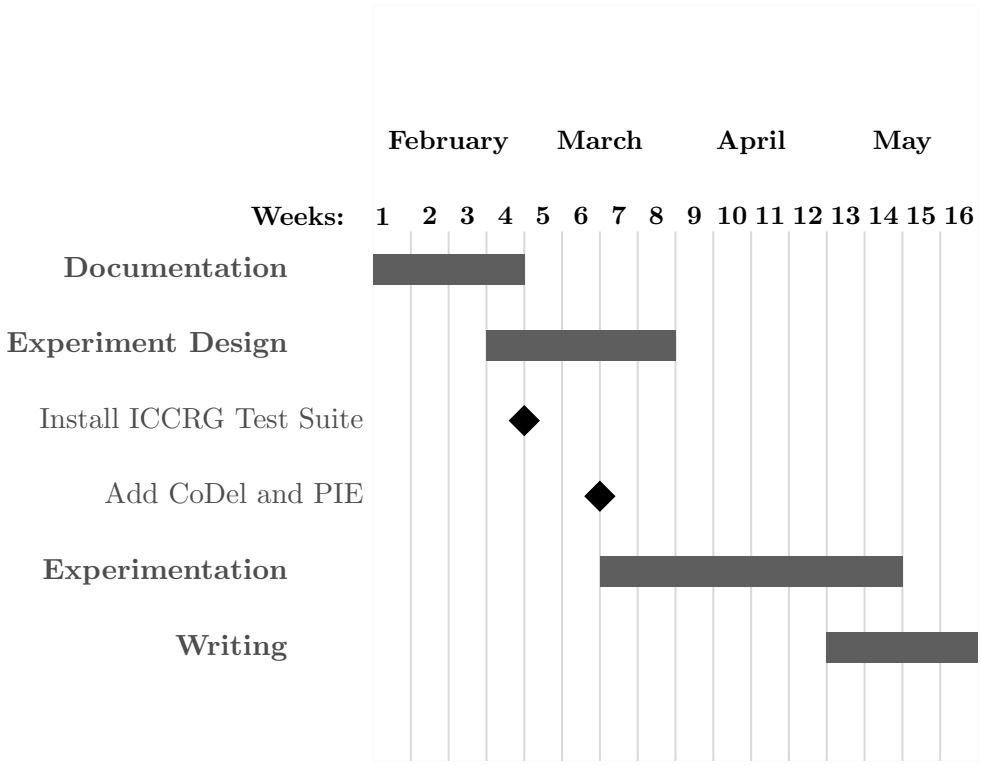
Our research could be spitted into four highly differentiated steps.

1. **Documentation:** On this step we studied the bibliography that covered this topic and research about the technology previously developed in order to address our goals. Once we decided to use the ICCRG TCP Evaluation Suite, we read the documentation about the suite in order to understand its capabilities.

2. **Experiment Design:** In this step, we installed the ICCRG test Suite on the server and started to get familiar to the environment. We performed short test simulations and wrote code to visualize and understand the results. We also added CoDel and PIE code into the suite in order to make tests with them.

3. **Experiment Simulation:** We started to make longer simulations, generate graphics about them and start making assumptions.

4. **Writing:** Although we generated some documentation since the beginning of the project, the last weeks were totally dedicated to write our findings.



A.2. Budget

1. Author:
   Sergio Maeso

2. Department:
   Telematic Engineering

3. Project Description:

Title:	AQM algorithms and their interaction with TCP congestion control mechanisms
Duration (months):	4
Indirect Costs Rate:	20%

4. Project's Total Budget (values in Euros):
   11.000 Euros

5. Budget Breakdown (Direct Costs):

STAFF						
Surname, Name	N.I.F.	Category	Dedication (Hours)	Cost per hour	Cost (Euros)	Signature
Campo, Celeste	N/A	Senior Engineer	40	60,00	2.400,00	
García, Carlos	N/A	Senior Engineer	40	60,00	2.400,00	
Maeso, Sergio	N/A	Junior Engineer	300	13,00	3.900,00	
		Total hours	380	**Total**	8.700,00	

EQUIPMENT					
Description	Cost (Euros)	% Usage in Project	Dedication (Months)	Depreciation Period (Months)	Chargeable Cost (Euros)
Intel Xeon Server	3.000,00	100	4	60	200,00
Linux Workstation	800,00	100	4	60	53,33
				Total	253,33

6. Costs Summary:

Direct Costs (Euros)	
Staff	8.700,00
Amortization	253,33
Task Externalization	0,00
Operation Costs	0,00
Indirect Costs (Euros)	1.311,00
Total (Euros)	10.264,33

This project's total budget add up to the amount of 11000 Euros.

Leganes, Month 7th, 2017

Project's Engineer,
(Signature)

Fdo. Sergio Maeso Jimenez

Bibliography

[1] R. Adams. Active queue management: A survey. *IEEE Communications Surveys Tutorials*, 15(3):1425–1476, Third 2013.

[2] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM, 2012 Proceedings IEEE*, pages 1620–1628. IEEE, 2012.

[3] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock. Host-to-host congestion control for tcp. *IEEE Communications Surveys Tutorials*, 12(3):304–342, Third 2010.

[4] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the performance of tcp pacing. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1157–1165. IEEE, 2000.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[6] Lachlan Andrew, Sally Floyd, and Gang Wang. Common tcp evaluation suite. Internet-Draft draft-irtf-tmrg-tests-02, IETF Secretariat, July 2009.

[7] G. Armitage. An experimental estimation of latency sensitivity in multiplayer quake 3. In *The 11th IEEE International Conference on Networks, 2003. ICON2003.*, pages 137–141, Sept 2003.

[8] S. Athuraliya, S.H. Low, V.H. Li, and Qinghe Yin Qinghe Yin. REM: active queue management. *IEEE Network*, 15(June):48–53, 2001.

[9] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):50, 2016.

[10] Neal Cardwell, Yuchung Cheng, Soheil Yeganeh, and Van Jacobson. Bbr congestion control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00, IETF Secretariat, July 2017. `http://www.ietf.org/internet-drafts/draft-cardwell-iccrg-bbr-congestion-control-00.txt`.

[11] Gwyn Chatranon, Miguel A Labrador, and Sujata Banerjee. A survey of tcp-friendly router-based aqm schemes. *Computer Communications*, 27(15):1424–1440, 2004.

[12] Shan Chen and Brahim Bensaou. Can high-speed networks survive with DropTail queues management? *Computer Networks*, 51(7):1763–1776, 2007.

[13] Yuchung Cheng, Neal Cardwell, Soheil Yeganeh, and Van Jacobson. Delivery rate estimation. Internet-Draft draft-cheng-iccrg-delivery-rate-estimation-00, IETF Secretariat, July 2017. `http://www.ietf.org/internet-drafts/draft-cheng-iccrg-delivery-rate-estimation-00.txt`.

[14] Sharon Choy, Bernard Wong, Gwendal Simon, and Catherine Rosenberg. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th annual workshop on network and systems support for games*, page 2. IEEE Press, 2012.

[15] Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis. Increasing tcp's initial window. Internet-Draft draft-ietf-tcpm-initcwnd-08, IETF Secretariat, February 2013.

[16] Mark Claypool and Kajal Claypool. Latency can kill: precision and deadline in online games. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 215–222. ACM, 2010.

[17] U.S. Federal Communications Commission. Voice telephone services: Status as of december 31, 2015. Technical report, Industry Analysis and Technology Division Wireline Competition Bureau, 10 2016.

[18] Declan Delaney, Tomás Ward, and Seamus McLoone. On Consistency and Network Latency in Distributed Interactive Applications: A Survey-Part I. *Presence: Teleoperators and Virtual Environments*, 15(2):218–234, 2006.

[19] Peter L Dordal. *An introduction to computer networks*. Online: http://intronetworks. cs. luc. edu, 2015.

[20] Nandita Dukkipati, Tiziana Refice, Yuchung Cheng, Jerry Chu, Tom Herbert, Amit Agarwal, Arvind Jain, and Natalia Sutin. An argument for increasing tcp's initial congestion window. *ACM SIGCOMM Computer Communications Review*, 40:27–33, 2010.

[21] Sonia Fahmy and Tapan Prem Karwa. Tcp congestion control: overview and survey of ongoing research. *Computer Science Technical Reports*, 2001.

[22] Gang Feng, A. K. Agarwal, A. Jayaraman, and Chee Kheong Siew. Modified RED gateways under bursty traffic. *IEEE Communications Letters*, 8(5):323–325, 2004.

[23] Wu-chang Feng, Kang G Shin, Dilip D Kandlur, and Debanjan Saha. The blue active queue management algorithms. *IEEE/ACM transactions on networking*, 10(4):513–528, 2002.

[24] Alessandro Finamore, Marco Mellia, Maurizio M Munafò, Ruben Torres, and Sanjay G Rao. Youtube everywhere: Impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 345–360. ACM, 2011.

[25] Sally Floyd, Ramakrishna Gummadi, Scott Shenker, et al. Adaptive red: An algorithm for increasing the robustness of red's active queue management, 2001.

[26] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993.

[27] Jim Gettys. Iw10 considered harmful. Internet-Draft draft-gettys-iw10-considered-harmful-00, IETF Secretariat, August 2011.

[28] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40:40–40:54, November 2011.

[29] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM Computer Communication Review*, volume 39, pages 51–62. ACM, 2009.

[30] Sangtae Ha, Yusung Kim, Long Le, Injong Rhee, and Lisong Xu. A step toward realistic performance evaluation of high-speed tcp variants. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks*, 2006.

[31] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: A new tcp-friendly high-speed tcp variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[32] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The newreno modification to tcp's fast recovery algorithm. RFC 6582, RFC Editor, April 2012.

[33] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of bbr congestion control. `https://doc.tm.uka.de/2017-kit-icnp-bbr-authors-copy.pdf`.

[34] Toke Hoeiland-Joergensen, Paul McKenney, dave.taht@gmail.com, Jim Gettys, and Eric Dumazet. The flowqueue-codel packet scheduler and active queue management algorithm. Internet-Draft draft-ietf-aqm-fq-codel-06, IETF Secretariat, March 2016. `http://www.ietf.org/internet-drafts/draft-ietf-aqm-fq-codel-06.txt`.

[35] C.V. Hollot, V. Misra, D. Towsley, and Wei-Bo Gong Wei-Bo Gong. On designing improved controllers for AQM routers supporting TCP\nflows. *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, 3:1–9, 2001.

[36] Common TCP Congestion Evaluation Suite. `https://trac.ietf.org/trac/irtf/wiki/ICCRG_tcpeval`.

[37] http://www.ikr.uni-stuttgart.de/Content/IKRSimLib/. Last accessed on May, 2017.

[38] V. Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, August 1988.

[39] V. Jacobson and R.T. Braden. Tcp extensions for long-delay paths. RFC 1072, RFC Editor, October 1988.

[40] OrÃ§un Kaya. High-frequency trading, Reaching the limits. *Automated Trader Magazine*, 41(5), 2016.

[41] Naeem Khademi, David Ros, and Michael Welzl. The new aqm kids on the block: an experimental evaluation of codel and pie. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 85–90. IEEE, 2014.

[42] Leonard Kleinrock. Power and deterministic rules of thumb for probabilistic problems in computer communications. In *Proceedings of the International Conference on Communications*, volume 43, pages 1–43, 1979.

[43] Chin-Fu Ku, Sao-Jie Chen, Jan-Ming Ho, and Ray-I Chang. Improving end-to-end performance by active queue management. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, volume 2, pages 337–340 vol.2, March 2005.

[44] Richard J La, Priya Ranjan, and Eyad H Abed. Analysis of adaptive random early detection (ared). *Teletraffic Science and Engineering*, 5:1271–1280, 2003.

[45] Fan Li, Jinsheng Sun, Moshe Zukerman, Zhengfei Liu, Qin Xu, Sammy Chan, Guanrong Chen, and King-Tim Ko. A comparative simulation study of TCP/AQM systems for evaluating the potential of neuron-based AQM schemes. *Journal of Network and Computer Applications*, 41:274–299, 2014.

[46] Steven H Low. Analytical methods for network congestion control. *Synthesis Lectures on Communication Networks*, 10(1):1–213, 2017.

[47] Mitchell JH Lum, Jacob Rosen, Hawkeye King, Diana CW Friedman, Thomas S Lendvay, Andrew S Wright, Mika N Sinanan, and Blake Hannaford. Teleoperation in surgical robotics–network latency effects on surgical performance. In *Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE*, pages 6860–6863. IEEE, 2009.

[48] Lefteris Mamatas, Tobias Harks, and Vassilis Tsaoussidis. Approaches to congestion control in packet networks. *Journal of Internet Engineering*, 1(1):22–33, 2007.

[49] Mininet Virtual Network. http://mininet.org/. Last accessed on July, 2017.

[50] Ministerio de Industria, Energía y Turismo. Orden iet/1090/2014, de 16 de junio, por la que se regulan las condiciones relativas a la calidad de servicio en la prestación de los servicios de comunicaciones electrónicas., 2014. http://www.boe.es/diario_boe/txt.php?id=BOE-A-2014-6729.

[51] Vishal Misra, Wei-Bo Gong, and Don Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. *SIGCOMM Comput. Commun. Rev.*, 30(4):151–160, August 2000.

[52] Dr. Kathleen M. Nichols, Van Jacobson, Andrew McGregor, and Janardhan Iyengar. Controlled Delay Active Queue Management. Internet-Draft draft-ietf-aqm-codel-07, Internet Engineering Task Force, March 2017. Work in Progress.

[53] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Communications of the ACM*, 55(7):42–50, 2012.

[54] Kathleen Nichols, Van Jacobson, Andrew McGregor, and Janardhan Iyengar. Controlled delay active queue management. Internet-Draft draft-ietf-aqm-codel-07, IETF Secretariat, March 2017. http://www.ietf.org/internet-drafts/draft-ietf-aqm-codel-07.txt.

[55] The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/. Last accessed on July, 2017.

[56] https://github.com/maesoser/ns2_bbr. Last accessed on April, 2017.

[57] Rong Pan, Preethi Natarajan, Fred Baker, and Greg White. Pie: A lightweight control scheme to address the bufferbloat problem. Internet-Draft draft-ietf-aqm-pie-10, IETF Secretariat, September 2016. http://www.ietf.org/internet-drafts/draft-ietf-aqm-pie-10.txt.

[58] Jon Postel et al. Transmission control protocol rfc 793, 1981.

[59] http://jupyter.org/. Last accessed on April, 2017.

[60] Hamzeh Qabaja and Marwan Bikdash. Performance comparison of autored under different tcp variations. In *SoutheastCon 2015*, pages 1–7. IEEE, 2015.

[61] SuperData research. Market brief - year in review 2016. Technical report, SuperData research, 2016. https://www.superdataresearch.com/market-data/market-brief-year-in-review.

[62] Injong Rhee. Cubic for fast long-distance networks. Internet-Draft draft-rhee-tcp-cubic-00, IETF Secretariat, March 2007. http://www.ietf.org/internet-drafts/draft-rhee-tcp-cubic-00.txt.

[63] Jacob Rosen, Blake Hannaford, and Richard M Satava. *Surgical robotics: systems applications and visions*. Springer Science & Business Media, 2011.

[64] Dario Rossi, Claudio Testa, and Silvio Valenti. Yes, we ledbat: Playing with the new bittorrent congestion control algorithm. In *PAM*, pages 31–40. Springer, 2010.

[65] Dario Rossi, Claudio Testa, Silvio Valenti, and Luca Muscariello. LEDBAT: The new BitTorrent congestion control protocol. In *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, 2010.

[66] Seungwan Ryu, Christopher Rump, and Chunming Qiao. Advances in active queue management (AQM) based TCP congestion control. *Telecommunication Systems*, 25(3-4):317–351, 2004.

[67] Jens Schwardmann, David Wagner, and Mirja Kühlewind. Evaluation of ared, codel and pie. In *Advances in Communication Networking*, pages 185–191. Springer, 2014.

[68] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low extra delay background transport (ledbat). RFC 6817, RFC Editor, December 2012.

[69] Madhavapeddi Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *ACM SIGCOMM Computer Communication Review*, volume 25, pages 231–242. ACM, 1995.

[70] W. Richard Stevens. Tcp slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. RFC 2001, RFC Editor, January 1997. `http://www.rfc-editor.org/rfc/rfc2001.txt`.

[71] K. Tan, J. Song, Q. Zhang, and M. Sridharan. Compound tcp: A scalable and tcp-friendly congestion control for high-speed networks. In *4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006*, May 2006.

[72] Niraj Tolia, David G Andersen, and Mahadev Satyanarayanan. Quantifying interactive user experience on thin clients. *Computer*, 39(3):46–52, 2006.

[73] `http://www.verizonenterprise.com/about/network/latency/`. Last accessed on May, 2017.

[74] Greg White. Active queue management in docsis 3.x cable modems. *CableLabs*, 2014.

[75] Wei Wu, Yong Ren, and Xiuming Shan. Stability analysis on active queue management algorithms in routers. In *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 125–132, 2001.

[76] Qin Xu, Fan Li, Jinsheng Sun, and Moshe Zukerman. A new tcp/aqm system analysis. *J. Netw. Comput. Appl.*, 57(C):43–60, November 2015.

[77] Yue-Dong Xu, Zhen-Yu Wang, and Hua Wang. {ARED}: a novel adaptive congestion controller. In *International Conference on Machine Learning and Cybernetics*, volume 2, pages 708–714, 2005.

[78] Takahiro Yasui, Yutaka Ishibashi, and Tomohito Ikedo. Influences of network latency and packet loss on consistency in networked racing games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, NetGames '05, pages 1–8, New York, NY, USA, 2005. ACM.

Acronyms

AIMD Additive increase/multiplicative decrease. 69

AQM Active Queue Management. 3, 15, 21–23, 25, 29, 34, 35, 38, 57, 61, 63, 69

ARED Adaptive Random Early Detection. 18, 22, 26, 34, 47, 48, 54, 57, 62, 69

BBR Bottleneck Bandwidth and RTT. 14, 15, 63, 69

BDP Bandwidth Delay Product. 8, 9, 15, 29, 30, 69

CoDel Controlled Delay. 1, 18–20, 22, 23, 26, 30–32, 34, 38, 41, 46–49, 54, 56, 57, 62, 69

CoDel-DT . 20, 34, 45, 47, 56, 57, 62, 69

CTCP Compound TCP. 3, 10–13, 26, 29–31, 34, 38, 40, 42, 48–50, 53, 54, 56, 62, 69

Cubic TCP Cubic. 3, 23, 26, 29, 49, 50, 53, 54, 56, 69

DNS Domain Name Server. 19, 69

DOCSIS Data over Cable Services. 62, 69

DRR Deficient Round Robin. 19, 69

ECN Explicit Congestion Notification. 16, 63, 69

EWMA Exponential Weighted Medium Average. 16, 17, 69

FIFO First In, First Out. 15, 69

fq-CoDel Fair Queuing CoDel. 19, 22, 26, 30–32, 38, 46, 47, 54, 56, 57, 62, 69

HTTP Hyper Text Transfer Protocol. 19, 69

ICCRG Internet Congestion Control Research Group. 2, 3, 23, 25–27, 69

IETF Internet Engineering Task Force. 3, 69

LEDBAT Low Extra Delay Background Transport. 12–14, 49, 63, 69

LFN Long Fat Networks. 9, 30, 34, 38, 69

LPCC Low Priority Congestion Control. 14, 69

MGT Misra, Gong and Towsley. 21, 69

MSS Maximum Segment Size. 15, 26, 69

MTU Maximum Transmission Unit. 19, 69

PIE Proportional Integral controller Enhanced. 1, 20, 22, 31, 32, 34, 38, 40–42, 47, 48, 54, 56, 57, 62, 69

RED Random Early Detection. 1, 16–18, 22, 23, 69

REM Random Exponential Marking. 17, 22, 26, 61, 69

RITE Reduce Internet Transport Latency. 3, 69

RTO Retransmission Timeout. 6, 7, 69

RTT Round Trip Time. 6, 7, 13–16, 23, 25, 29, 32, 37, 39, 40, 47, 48, 53, 54, 62, 69

TCP Transport Control Protocol. 1, 23, 34, 38, 49, 54, 56, 61–63, 69